Rahmenbedingungen für erfolgreiches Software-Reuse

E. Hochmüller, R. Mittermeir

Institut für Informatik
Universität Klagenfurt
Universitätsstr. 65-67
A-9022 Klagenfurt
e-mail: {elke,mittermeir}@ifi.uni-klu.ac.at

ABSTRACT

Die Einführung eines systematischen Konzeptes zur Wiederverwendung von Software wird heute in vielen Softwarehäusern als zentrale Strategiefrage angesehen. Allerdings stellen sich der erfolgreichen Umsetzung vorhandener Technologien eine Fülle von Barrieren entgegen, die oft zu überraschenden negativen Effekten führen.

In dieser Arbeit wird gezeigt, daß in der Praxis erfolgreiches Software-Reuse einen ganzheitlichen Ansatz erfordert. Dieser beginnt mit einer Überprüfung der Produktstruktur und führt über Veränderungen in der Prozeß- und Organisationsstruktur bis hin zu einer belebenden Reform der Motivationsstruktur. Letztere sollte sowohl quantitativ-kostenrechnerische als auch psychologische Aspekte berücksichtigen.

1. Motivation und Problemumfeld

Wegen wachsender Anforderungen aus nahezu allen Anwendungsgebieten einerseits und technischen wie ökonomischen Entwicklungen im Hardware-Bereich andererseits steht die Software-Entwicklung unter sehr starkem Produktivitäts- und Qualitätsdruck. Durch zusehends stärkeren Einsatz von CASE-Tools wird versucht, diese Herausforderung aufzunehmen. Allerdings ist völlig klar, daß CASE lediglich inkrementelle Verbesserungen der Situation bringen kann. Somit liegt auf der Wiederverwendung von bereits entwickelter Software in neuen Projekten sehr hoher Erwartungsdruck. Dieser hat sich - trotz intensiver Bemühungen in Wissenschaft und Wirtschaft - in den letzten Jahren freilich nur partiell erfüllen lassen. Die Gründe dafür scheinen aus heutiger Sicht weniger technischer, sondern vielmehr administrativer Natur zu sein. In der vorliegenden Arbeit sollen diese Aspekte aufgezeigt werden, um so jenen, die systematische Software-Wiederverwendung in ihrem Unternehmen einführen wollen, Anregungen zu bieten, wie diese Hürden, die nur zu oft übersehen werden, aus dem Weg geräumt werden können.

Um einen möglichst umfassenden Überblick über derartige nicht-technische Einflußfaktoren zu geben, die oft bis in die Marketing-Strategie eines Software-Hauses reichen, beginnt diese Arbeit mit einer Darstellung der "environmental factors", innerhalb derer die Software-Entwicklung in einem Unternehmen abläuft und die bei Entwicklung einer Reuse-Strategie zu beachten sind. Dieses Kernkapitel der Arbeit gliedert sich in die Aspekte Produktstruktur, Prozeßstruktur, Organisationsstruktur und Motivationsstruktur. Im Anschluß an diese Diskussion unterschiedlicher Facetten eines funktionierenden Reuse-Konzeptes werden Aspekte des Wandels von konventioneller Software-Entwicklung auf reuse-orientierte Software-Entwicklung gezeigt.

2. Erfahrungen mit "environmental factors" für Software-Reuse

Im allgemeinen werden in der Literatur zur Software-Wiederverwendung lediglich relativ allgemein gehaltene Lösungskonzepte vorgestellt, oder es werden Werkzeuge und Methoden präsentiert, die scheinbar universell anwendbar sind. Demgegenüber gibt es nach unserer Einschätzung bloß einige Problembereiche, die unabhängig von der konkreten Ausgangslage

auftreten, wenn Software-Reuse institutuionalisiert werden soll. Insbesondere für Überleitungsaspekte mag dies gelten. Die angestrebte Zielsituation wird jedoch sicherlich einige Randbedingungen berücksichtigen müssen, um nicht entweder durch Nichterreichung des Unerreichbaren oder durch Ineffizienz eines überzogenen Konzeptes zu Frustration zu führen und so einen grundsätzlich guten Ansatz zum Scheitern zu zwingen.

Um die unterschiedlichen Facetten, unter denen Software-Entwicklung stattfindet einigermaßen allgemein diskutieren zu können, versuchen wir die Rahmenbedingungen der Software-Entwicklung an vier Grunddimensionen festzubinden. Es sind dies:

- **was** wird entwickelt: *Produktstruktur*,
- **wie** wird entwickelt: *Prozeßstruktur*,
- **womit** wird entwickelt: *Organisationsstruktur*,
- **warum** bemüht sich der einzelne: *Motivationsstrukur*.

Im folgenden werden diese einzelnen Dimensionen näher beleuchtet.

2.1. Produktstruktur

Die Produktstruktur eines Software-Hauses ist zweifellos von entscheidendem Einfluß auf die zu verfolgende Wiederverwendungs-Strategie. Um hier eine erste grobe Unterscheidung zu treffen, wird zwischen

- Software-Häusern, die am freien Markt agieren, und
- in-Haus Software-Entwicklungsabteilungen

zu differieren sein.

Diese Unterscheidung wird insbesondere deshalb wesentlich, weil das Software-Haus wohl mehr Flexibilität in seiner Produktpalette hat, als die grundsätzlich an Unternehmenszielen und Unternehmensprodukten der Trägerorganisation orientierte Entwicklungsabteilung. Andererseits wird letztere ihr mittel- und langfristiges Tätigkeitsspektrum sicherlich viel klarer einschätzen können als die Mehrzahl von Software-Häusern. Da freilich das Gesamtspektrum der Tätigkeitsfelder von Software-Häusern produktseitig jenes von in-Haus-Software-Entwicklern umschließt, werden wir in den folgenden Ausführungen primär auf erstere abheben. In-Haus-Entwickler mögen sich anhand der dort gegebenen Gliederung zuordnen.

Als Produktstrategien von Software-Häusern wollen wir idealtypisch folgende Kategorien näher betrachten:

a) Entwicklung kundenspezifischer Spezialsoftware ("*Einzelprodukte*"),
b) Entwicklung und Wartung einiger weniger Spezialsysteme für ein spezifisches Marktsegment ("*Produkt von der Stange*"),
c) Entwicklung und Wartung von *Software-Paketen*, die in unterschiedlicher Form an unterschiedliche Kundengruppen oder Marktsegmente ausgeliefert werden,
d) "*turn-key systems*", i.e. Gesamtlösungen, die neben selbst entwickelter Software auch (ggf. sogar dominant) zugekaufte Hard- und Software-Komponenten enthalten.

Bei den Entwicklern von *Einzelprodukten* wird man davon ausgehen dürfen, daß sie sich auf ein spezifisches Marktsegment spezialisiert haben. Innerhalb eines solchen wird es immer wieder relativ komplexe Komponenten geben, die entweder komplett oder in unterschiedlichen Varianten in Produkte für unterschiedliche Kunden integriert werden können. Diese entstehen freilich nicht "von selbst", sondern es wird bereits in der Phase der Auftragsannahme darauf Bedacht zu nehmen sein, wie weit ein zu erstellendes Produkt wirklich "neu" zu entwickeln ist und wie weit man sich auf Synergie-Effekte zu parallel laufenden oder zu bereits abgeschlossenen Projekten stützen kann. Das Wiederverwendungspotential wird sich hier somit vor allem auf der Ebene relativ komplexer Teilsysteme bewegen, was eher umfassende Anforderungen an die Beschreibung der einzelnen Komponenten stellen wird. Da derartige Software-Entwickler mitunter entsprechend vorgegebener Kundenforderungen in unterschiedlichen Programmiersprachen und für unterschied-

liche Hard- und Software-Plattformen arbeiten, stellt sich in diesem Fall auch sehr oft die Frage nach dem bevorzugten Abstraktionsgrad für wieder zu verwendende Komponenten. Dieser wird sehr oft nicht auf Code-Ebene liegen, sondern auf einer dieser vorgelagerten Ebene des Entwurfes oder eines sonstigen "upstream products".

Im Falle der Entwicklung von *Produkten von der Stange* für einen wohldefinierten aber noch nicht auf Kundenebene bekannten Markt sollten - so wie im Fall des Einzelproduktes in den Vertragsabschluß - Reuse-Überlegungen bereits in die Überlegungen zur Gestaltung der Produkt-Palette einfließen. Auch dies setzt voraus, daß Reuse-Überlegungen nicht auf Code-Ebene ansetzen, sondern daß bereits zu einem frühen Zeitpunkt und auf sehr hohem Abstraktionsniveau die Frage nach einer effizienten Entwicklung unter Rückgriff auf Synergie-Effekte gestellt werden muß. Weiters wird hier die Frage der Fortentwicklung einer Produktlinie spezielles Augenmerk verdienen. Diese Thematik wird zwar im allgemeinen unter dem Thema "Wartung" behandelt, im Sinne von Basili [Basi90] sollte jedoch gerade in diesen Fällen die Verbindung von Wartung und Wiederverwendung in den Vordergrund gerückt werden.

Ähnliches gilt für *Software-Pakete*. Allerdings muß hier noch viel stärker als bei "Stangen-Ware" davon ausgegangen werden, daß diese auf unterschiedlichen Plattformen in unterschiedlichen Varianten und Versionen laufen. Die Frage systematischer Software-Wiederverwendung ist dabei nicht nur für effiziente Entwicklung sondern auch für effiziente Wartungsunterstützung entscheidend. Durch entsprechendes "Reuse with Modification" können sich in diesen Fällen nicht unbeträchtliche Erleichterungen für das Konfigurations-Management ergeben. Allerdings wird gerade im Wartungsbereich die Frage zu lösen sein, ob die wiederverwendeten Komponenten im Sinne einer View-Semantik (automatisches Nachführen von Änderungen an der Grundkomponente) oder im Sinne einer Snapshot-Semantik (Trennung der Beziehung zwischen Mutter- und Tochter-Komponente) erfolgen sollte.

Der Begriff "*turn-key systems*" kann wohl sehr weit gefasst werden. Aus der Sicht dieser Diskussion wollen wir jedoch vor allem auf die dabei sehr oft entstehende Schnittstellen-Problematik abheben. Wenn Software-Entwickler an eine extrem hohe Zahl fester Schnittstellen gebunden sind, reduziert dies selbstverständlich die Freiheitsgrade des Entwurfes und somit auch das Reuse-Potential. Entwicklungs-Manager werden sich in solchen Situationen sicherlich nach dem erzielbaren Anspruchsniveau für Reuse fragen müssen. Mitunter wird dieses in der Verwendung relativ kleiner, vorcodierter Service-Routinen bereits das ökonomisch sinnvoll erreichbare Optimum finden.

Wenn sich in-Haus-Software-Entwickler sicherlich einer (meist mehrerer) dieser für Software-Häuser postulierten Kategorien zuordnen können, ergibt sich für sie jedoch noch eine andere, scheinbar triviale, in der Praxis jedoch schwierig zu lösende Problematik: jene der Kostenzurechnung und des fairen Preises.

Im Detail wird auf die Kostenfrage in der Überleitungs-Diskussion nochmals einzugehen sein. Es soll jedoch bereits hier festgehalten werden, daß für das Software-Haus die Frage einer adäquaten Kostenrechnung wohl so zu lösen sein wird, daß man die mit der Erstellung und Pflege der Reuse-Bibliothek verbundenen Kosten als Investitions- (und Gemein-)kosten zu sehen hat und die Kosten der Verwendung (Suche und Adaptierung) wohl projektspezifisch zuzurechnen sein werden. Die Kostendifferenz (und somit der entsprechende Reuse-Gewinn) zwischen kompletter Neuentwicklung und Entwicklung mit Reuse darf als Deckungsbeitrag zu den Investitionskosten aufgefaßt werden. Nahezu kurioserweise stößt diese unmittelbar einsichtige Lösung bei einigen In-Haus-Entwicklern aus Günden der "Kostenwahrheit" auf Schwierigkeiten. Eine dazu verwandte, jedoch inverse Problematik kann sich aus rechtlichen Gründen ergeben und ist bei der Vertragsgestaltung durch das Software-Haus zu berücksichtigen.

Obige Diskussion zeigte, daß bei Einführung einer Reuse-Strategie zur Vermeidung späterer Enttäuschungen zu berücksichtigen sein wird, was man zum Gegenstand der Wiederverwendung machen möchte (Code, Entwürfe, Domain-Modelle, Prototypen, Anforderungsdokumente, Verträge, Dokumentation, ...) und auf welche Reuse-Granularität (wohl definierte kleine Service-Routine oder

eng umschriebener Service-Modul, Komponenten aus dem Anwendungsbereich oder ggf. ganze Subsysteme) man zielen sollte.

Bei dieser Grundsatz-Entscheidung ist die Frage nach der verwendeten Entwicklungsmethode oder nach der eingesetzten Programmiersprache relativ sekundär. Objektorientierung, und die damit verbundene Vererbung, ist zweifellos hilfreich. Sie ist insbesondere für Fragen, die am Schnittpunkt zwischen Systemevolution und Wiederverwendung stehen, nützlich, stellt jedoch weder eine conditio sine qua non noch einen taxfreien Lösungsautomatismus für Reuse-Fragen dar.

2.2. Prozeßstruktur

Die Struktur des Software-Entwicklungsprozesses ist ein weiterer "environmental factor", dem jedoch im Gegensatz zur Produktstruktur bisher zu wenig Bedeutung beigemessen wurde. Erst seit wenigen Jahren gibt es diesbezügliche Forschungsaktivitäten [Oste87], doch ist der Software-Prozess seither Gegenstand intensiver Forschung geworden (siehe entspr. IEEE-Workshops und ESPRIT-Projekte). Speziell auf Wiederverwendungsaspekte wurde in [Prie91b], [Basi91], [Börs91] jedoch auch in [Hend 90] eingegangen. Bevor wir jedoch unser Modell des Wiederverwendungsprozesses vorschlagen, gehen wir kurz auf Formen und Reifegrad des allgemeinen Software-Entwicklungsprozesses ein.

a) Formen des Prozesses

Für den herkömmlichen Software-Entwicklungsprozeß hat sich wohl das klassische Wasserfallmodell [Boeh81] als de-facto-Standard durchgesetzt, das jedoch aufgrund unterschiedlicher Gliederung von Phasen und Meilensteinen in verschiedensten Varianten angewandt wird. Andere Alternativen finden sich in Meta-Modellen, wie z.B. im Spiralmodell [Boeh88], das Software-Entwicklung und Risikomanagement miteinander in Beziehung setzt. Prototyping ist ein weiterer Ansatz, der mehr Freiräume mit sich bringt, da keine Einteilung in Phasen verlangt und keine Erreichung von Meilensteinen vorgeschrieben werden und rascheres Feedback erzielt werden kann [Floy84], [Bisc89]. Für Großprojekte wurden unter expliziter Berücksichtigung des Wiederverwendungs-Aspektes auch bereits zweistufige Prozeßmodelle vorgeschlagen [Mitt 91].

b) Reifegrad des Prozesses

Unbeschadet der konkreten Gestaltung des Software-Entwickungsprozesses lassen sich bei unterschiedlichen Software-Entwicklern unterschiedliche Reifegrade dieses Prozesses feststellen [Hump 89], [Yeh 91], [Mitt92].

Dabei mag auf den ersten Blick verwundern, daß trotz der heftigen Diskussion zur Software-Wiederverwendbarkeit die Frage nach der Existenz einer konsistenten Wiederverwendungsstrategie erst auf Stufe 3 ("defined process") der fünfstufigen Skala gestellt wird. Geht man dem Wesen systematischer Wiederverwendung jedoch auf den Grund, erkennt man bald, daß diese nur dann angenommen wird, wenn innerhalb des zu lösenden Problembereiches auch namhafte Anteile mittels Wiederverwendung von Teilen oder Konzepten abgedeckt werden können. Dies setzt freilich voraus, daß das im Problem- bzw. Lösungsbereich enthaltene Reuse-Potential sowohl im Sinne des nachstehend behandelten Development for Reuse als auch des Development with Reuse genutzt wird. Beides ist freilich nur möglich, wenn Software nach entsprechenden Standards entwickelt wird.

Die Einhaltung von Standards setzt allerdings voraus, daß Software-Entwicklung nicht bloß als Kodierung verstanden wird, sondern daß auch die "Produkte" früher Phasen, wie Anforderungsdokumente, Spezifikationen, Entwürfe, als wertvolle (wiederverwendbare) Zwischenprodukte angesehen werden. Nur so kann durch entsprechende Qualitätssicherungsmaßnahmen erreicht werden, daß aus dem normalen Betrieb heraus wiederverwendbare Komponenten entstehen und daß sich

die einzelnen Entwickler zur Überwindung aller psychologischen und sonstigen Barrieren aufraffen und bereitgestellte Zwischenprodukte auch tatsächlich in ihrer Problemlösung verwenden.

c) spezifische Wiederverwendungselemente im Software-Prozeß

Zur Unterstützung von Software-Wiederverwendung werden in der Fachliteratur vor allem folgende Bereiche behandelt: *Domain Analysis*, *Reverse Engineering for Reuse*, *Development for Reuse* und *Development with Reuse*.

Unter *Domain Analysis* [Prie91a] werden Aktivitäten subsummiert, die zu einem besseren Verständnis des Anwendungsbereiches beitragen. Für Software Reuse bedeutet dies, daß Komponenten (bzw. Konzepte), die sich während der Domain Analysis herauskristallisiert haben, ein erhöhtes Reuse-Potential aufweisen. Durch gezielte Analyse der Anforderungen eines bestimmten Anwendungsbereiches können generische Komponenten vorbereitet werden, und dadurch kann bereits auf zukünftige Erfordernisse Bedacht genommen werden. Somit können potentiell wiederverwendbare Komponenten und deren mögliche Kombinationsalternativen identifiziert werden. Diese Komponenten sind besonders zur Wiederverwendung geeignet, da sie die wichtigsten im gesamten Anwendungsbereich erforderlichen Funktionalitäten berücksichtigen und somit leicher in neue Systeme integrierbar sind.

Die Hauptaufgabe von Reverse Engineering ist die Transformation einer in Form von Source-Code vorhandenen Darstellung eines Software-Systems in eine Repräsentation dieses Systems auf einem höheren Abstraktionsniveau. *Reverse Engineering for Reuse* befaßt sich mit der Suche nach Code, der in potentiell wiederverwendbare Komponenten eingebunden werden soll [Gall92]. Die notwendige Beschreibung auf höherem Abstraktionsniveau wird - wie beim herkömmlichen Reverse Engineering - aus existierender Software extrahiert, wobei zusätzlich Transformationen zur Codeverbesserung und Generalisierungen zur Erreichung der allgemeinen Anwendbarkeit der somit "neu" generierten wiederverwendbaren Software-Komponenten notwendig sind.

Development for Reuse [Dusi89] beschäftigt sich mit der Bildung von neuen Software-Komponenten zur späteren Wiederverwendung. Zum Unterschied zu herkömmlichen "guten" Software-Engineering-Praktiken (z.B. Structured Analysis/Structured Design, strukturierte Programmierung) handelt es sich hier nicht nur um die Entwicklung von leicht wartbarer Software, sondern es muß zum einen bereits auf zukünftige Anforderungen Bedacht genommen werden, und zum anderen muß die Unabhängigkeit der einzelnen Entwurfseinheiten gewährleistet werden. Somit muß insbesondere darauf geachtet werden, daß einerseits die jeweilige Software-Komponente so generisch wie möglich ist, um durch geeignete Instanzierungen oftmalige Wiederverwendung zu gewährleisten, andererseits muß dazu die Komponente auch in bezug auf ihr unmittelbares Umfeld unabhängig sein, um in vielen verschiedenen Kontexten einsetzbar zu sein. Unter den bekannten Entwurfsmethoden unterstützen objektorientierte Design-Methoden besonders die Bildung von wiederverwendbaren Komponenten. Weiters ist Standardisierung von Software (z.B. Software-Normalisierung [Mitt90]) eine vielversprechende Strategie zur Entwicklung von wiederverwendbaren Komponenten. Außerdem können die Verwendung von abstrakten Datentypen und das Konzept der Parametrisierung (z.B. Prozedurvariablen, Generics) die potentielle Wiederverwendbarkeit sehr erhöhen.

Im Zuge von *Development with Reuse* werden die durch *Development for Reuse* und *Reverse Engineering* bereitgestellten Komponenten wiederverwendet, indem zusätzlich zum Auffinden von gesuchten Software-Komponenten auch für die Kombination dieser Einheiten Sorge getragen wird. Falls gefundene Komponenten nur teilweise den gestellten Anforderungen genügen, müssen entsprechende Ergänzungen vorgenommen werden.

Zur Software-Entwicklung unter Verwendung von vorgefertigten Komponenten ist es vor allem notwendig, daß Software Reuse auch im Software Life Cycle gezielt berücksichtigt wird und daß durch entsprechende Strategien für die Adaption und das Zusammenwirken von wiederverwendbaren Komponenten gesorgt wird [Trac90]. Von besonderer Bedeutung ist eine geeignete Klassifi-

kation von Software-Komponenten, um das Auffinden gesuchter bzw. ähnlicher Komponenten zu gewährleisten. Außerdem muß besonderes Augenmerk auf die Pflege von wiederverwendbaren Komponenten (Restrukturierung, Reorganisation) gelegt werden. In diesem Zusammenhang wird klar, daß nicht nur vollständige Software-Systeme sondern auch Subsysteme bzw. wiederverwendbare Komponenten einem Lebenszyklus unterworfen sind.

d) Reuse-Prozeßmodell

Das im folgenden vorgeschlagene Prozeßmodell dient zur Berücksichtigung des Einsatzes einer Bibliothek wiederverwendbarer Software-Komponenten in Entwicklungsprojekten, wobei die Methode der Entwicklung der eigentlichen Projektsoftware so allgemein wie möglich gehalten wird, um keine zu einschränkende Lösung zu bieten.

Im wesentlichen bedingt reuse-oriented Software Engineering zwei verschiedene Prozesse, die in die jeweils gewählte Entwicklungsmethodologie von Software eingebunden werden müssen (vgl. Abb. 1). Einerseits handelt es sich um den Prozeß der Entwicklung von wiederverwendbaren Komponenten zur Aufnahme in die Software-Bibliothek (*Development for Reuse*), andererseits muß die bisher übliche Entwicklungsstrategie von Projektsoftware mit dem Prozeß der Suche und Verwendung von Komponenten, die in der Software-Bibliothek abgelegt sind, ergänzt werden (*Development with Reuse*). Eine Kombination dieser zwei Prozesse ist natürlich nur dann möglich, wenn Software aus den einzelnen Entwicklungsprojekten wenigstens teilweise in die Software-Bibliothek aufgenommen wird [Hoch92]. Somit wird die Bibliothek einerseits aus spezieller Auftragsfertigung für diese Reuse-Library und andererseits aus kundenbezogenen Entwicklungsprojekten gespeist.

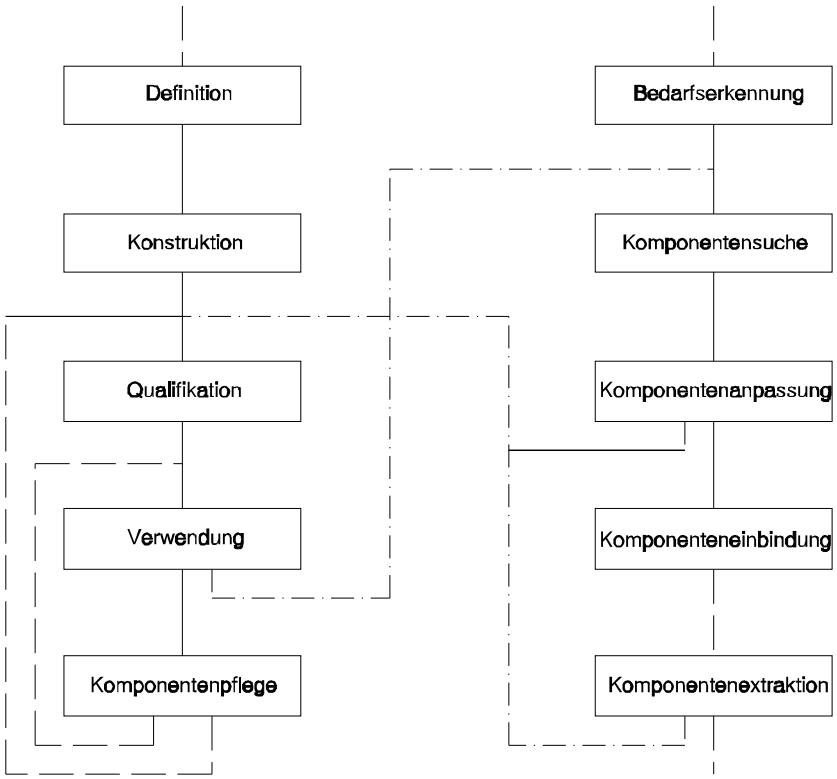

Abb. 1: Reuse-Prozeßmodell

Entwicklung von wiederverwendbaren Komponenten

Der Prozeß der *Entwicklung von wiederverwendbaren Komponenten* (*Development for Reuse*) zur Ablage in der Software-Bibliothek kann im wesentlichen in 5 Schritte eingeteilt werden, die im linken Teil der Abb. 1 dargestellt sind:

- *Definitionsphase*: Die Nachfrage nach einer bestimmten wiederverwendbaren Komponente wird registriert. Die Entwicklungskosten werden veranschlagt. Der potentielle Nutzen wird gegen diese Kosten abgewogen, und eine entsprechende Entwicklungsentscheidung wird getroffen.

- *Konstruktionsphase*: Die Komponente wird durch die Projektgruppe entworfen und anschließend implementiert. Dabei werden festgelegte Standards eingehalten.

- *Qualifikationsphase*: Die Komponente wird vom Verwalter der Software-Blibliothek überprüft. Notwendige Änderungen werden vollzogen. Am Ende der *Qualifikationsphase* erfolgt zur Archivierung und Such-Unterstützung der Komponente ihre Klassifikation.

- *Verwendungsphase*: Die überprüfte und klassifizierte Komponente steht nach Aufnahme in die Software-Bibliothek zur weiteren Verwendung zur Verfügung. Die in der *Verwendungsphase* befindlichen Komponenten können zur *Entwicklung mit wiederverwendbaren Komponenten* (*Development with Reuse*) herangezogen werden.

- *Komponentenpflege*: Diese Phase ist notwendig, um zu gewährleisten, daß eine Software-Bibliothek nicht nur wächst, sondern auch in regelmäßigen Zeitabständen gewissen Restrukturierungsmaßnahmen unterzogen wird. Diese sind notwendig, um die Qualität der Bibliothek zu erhalten bzw. zu verbessern. In diesem Zusammenhang kommt dem Verwalter der Bibliothek besondere Bedeutung zu. Auf seine Rolle wird im Rahmen der Organisationsstruktur (Abschnitt 2.3.) noch genauer eingegangen.

  Falls durch diese Restrukturierung Änderungen an einer Komponente erforderlich geworden sind, muß diese wiederum die *Qualifikationsphase* durchlaufen, um in veränderter Gestalt in die Bibliothek wiederaufgenommen zu werden.

  Betrifft die Restrukturierung ausschließlich das Klassifikationsschema der Bibliothek, ohne Änderung der Komponente selbst, so erfolgt in dieser Phase die Klassifikation der Software-Komponente nach dem neuen Schema; anschließend wird die Komponente in die *Verwendungsphase* zurückgereicht.

Entwicklung mit wiederverwendbaren Komponenten

Bei der Entwicklung von Software während eines Projektes betrifft ein Teilprozeß die *Entwicklung mit wiederverwendbaren Komponenten* (*Development with Reuse*). Dieser Prozeß (rechter Teil in Abb. 1) wird folgendermaßen in den gewählten Entwicklungsprozeß eingebunden und gegliedert:

- *Bedarfserkennung*: Zu einem bestimmten Zeitpunkt im umschließenden Software-Entwicklungsprozeß - meist im Anschluß an die Spezifikationsphase bzw. in der Entwurfsphase - wird erkannt, welche Komponenten für die Erstellung des Software-Systems benötigt werden.

- *Komponentensuche*: In der Bibliothek wird nach geeigneten Komponenten gesucht, die der Spezifikation der gesuchten Komponente entsprechen. Ein spezielles Suchverfahren wird zu diesem Zwecke angewandt.

- *Komponentenanpassung*: Je nach Erfolg der vorigen Phase bieten sich die folgenden drei unterschiedlichen Vorgangsweisen an.

- Wurde bei der Komponentensuche eine passende Komponente gefunden, kann diese ohne Veränderung übernommen werden. Gleiches gilt für eine gefundene Komponente, die zwar der geforderten Spezifikation nicht gänzlich entspricht, jedoch soweit annehmbar ist, daß die ursprünglichen Anforderungen im Lichte der gefundenen Komponente abgeändert werden können.

- Wurde bei der Suche eine Komponente gefunden, die zwar einige Anforderungen erfüllt, jedoch nicht vollständig der Spezifikation entspricht, muß diese Komponente weiteren *funktionalen Änderungen* unterzogen werden.

- Wurde keine geeignete Komponente gefunden, muß eine *völlige Neuentwicklung* erfolgen.

Die beiden letzteren Fälle bieten einen Anknüpfungspunkt für die Kombination mit der *Entwicklung für Reuse* (*Development for Reuse*), indem die so entstandenen Komponenten selbst wiederum in die Software-Bibliothek aufgenommen werden können (Abb. 1).

- *Komponenteneinbindung*: In dieser Entwicklungsstufe erfolgt die Einbettung der während der vorangegangenen Phasen gewonnenen Komponenten in ihre spezielle Umgebung. Dabei handelt es sich im wesentlichen um *anwendungsspezifische Parametrisierung*.

- *Komponentenextraktion*: Es liegt in dieser Phase bereits das vollständige Anwendungssystem vor. An dieser Stelle ist es nun möglich, weitere Komponenten zur Aufnahme in die Software-Bibliothek zu extrahieren. Dabei handelt es sich im Gegensatz zu den *funktionsspezifischen* Ergebnissen der *Komponentenanpassungsphase* um *anwendungsspezifische* Komponenten.

  Zwischen dem Entwicklungsschritt der *Komponentenextraktion* und der vorliegenden Phase können weitere Entwicklungsstadien liegen, die dem umschließenden Software-Entwicklungsprozeß zugerechnet werden können. Dieser Umstand soll durch die strichlierte Linie in Abb.1 (DwR) ausgedrückt werden.

Zusammenwirken beider Prozesse

Da beide vorgestellten Entwicklungsprozesse auf die gleiche Software-Bibliothek zugreifen, bestehen einige Berührungspunkte (Abb. 1).

Zum einen besteht eine Querverbindung von der *Verwendungsphase* zur Phase der *Komponentensuche*. Weiters besteht im Falle der *Komponentenanpassung* (Änderung bzw. Neuentwicklung) ein Anknüpfungspunkt zur *Qualifikationsphase*, um zu prüfen, ob die geänderte bzw. neue Komponente in die Software-Bibliothek aufgenommen werden kann. Schließlich ist im Anschluß an eine erfolgte *Extraktion einer anwendungsspezifischen Komponente* ihre *Qualifikation* zur Ablage in der Bibliothek erforderlich.

Dabei soll darauf hingewiesen werden, daß die Frage, wie hoch der Qualifikations- und Zertifikationsaufwand beim Einbringen einer Komponente sein sollte und welchen Prüfaufwand der Entnehmer leisten muß, unternehmensspezifisch zu regeln sein wird. Jedenfalls sollte jede Komponente aber einen Zertifikationsindikator tragen.

2.3. Organisationsstruktur

Die vorstehend beschriebenen zueinander asynchron ablaufenden Entwicklungsprozesse werfen in Unternehmen, die nach konventioneller Projektorganisation strukturiert sind, im allgemeinen erhebliche Probleme auf. Zu stark sind die Einwirkungen von Termindruck und sonstigen Streßfaktoren, um im Rahmen klassischer Projekte wiederverwendbare Software erstellen und pflegen zu können.

Um diese Problematik in den Griff zu bekommen, ist es sinnvoll, neben der regulären Projektorganisation eine eigene Reuse-Gruppe zu installieren, die die einzelnen Projekte sowohl in bezug auf die Wiederverwendung von Vorhandenem als auch auf die Wiederverwendbarmachung von neu zu Entwickelndem berät. Im folgenden werden Aspekte dieser Reuse-Gruppe und ihres Zusammenwirkens mit den Projektgruppen diskutiert.

a) Software-Bibliothek

Von besonderer Bedeutung ist die Einrichtung und Verwendung einer unternehmensweiten Software-Bibliothek. Der Inhalt dieser Bibliothek soll nicht nur auf Code-Niveau beschränkt sein; auch Entwürfe und freitextliche Dokumentationen sollen Aufnahme finden. Verbindungen zwischen den einzelnen Entwürfen und ihren realisierten Implementierungen sollten herstellbar sein.

Es muß darauf geachtet werden, daß nicht jedes beliebige Stück Code in die Bibliothek aufgenommen wird. Schließlich ist die Größe einer Software-Bibliothek kein Garant für erfolgreiches Reuse. Entscheidend ist vielmehr das Zusammenwirken aus Umfang, Qualität und Einfachheit des Auffindens gesuchter Komponenten. Außerdem kann die "Umschlags-"Häufigkeit einer Komponente ein wichtiger Indikator der Sortimentspflege sein.

Im Zusammenhang mit der Bestückung der Software-Bibliothek sind also eigene Prüfprozesse von besonderer Bedeutung. Dabei sind vor allem zwei Arten von Komponenten wünschenswert: solche, die zwar hoch spezifisch sind, aber deren Wahrscheinlichkeit der "*as-it-is*"-Wiederverwendung entsprechend hoch ist, und solche, die so allgemein gehalten sind, um durch Parametrisierung bzw. Instanzierung an viele weitere Fälle angepaßt werden können. Durch die erwähnten Überprüfungen soll nun gewährleistet werden, daß wirklich nur Software-Komponenten mit eher hohem Reuse-Potential in der Bibliothek Aufnahme finden. Meist kann eine Komponente nur nach entsprechender vorheriger Abänderung in die Bibliothek aufgenommen werden.

Zur rascheren Auffindung der einzelnen Komponenten sollen diese in eine adäquate Index- bzw. Klassifikationsstruktur eingebettet werden. Eventuelle Werkzeugunterstützung von Ablage und Suche wäre wünschenswert.

b) Reuse-Gruppe und Software-Bibliothekar

Durch Institutionalisierung einer projektunabhängigen Reuse-Gruppe soll die Praktizierung von Software-Wiederverwendung forciert werden. Aufgaben dieser Reuse-Gruppe sind die Beratung der einzelnen Projektgruppen zur Entwicklung von und mit wiederverwendbaren Komponenten, die Durchführung von reuse-spezifischen Schulungen der Projektmitarbeiter und die Verwaltung der Reuse-Bibliothek. Mit letzterer Funktion sollte ein Mitglied der Reuse-Gruppe betraut werden, das die Rolle eines Software-Bibliothekars übernimmt. Da die Mächtigkeit einer Software-Bibliothek sehr stark von ihrem Inhalt abhängig ist, ist insbesondere diese Rolle mit hoher Verantwortung verbunden.

Folgende Aufgaben sind vom Software-Bibliothekar zu erfüllen:

- Entscheidung über in die Bibliothek aufzunehmende Komponenten
- Pflege der Bibliothek:
  - Klassifikation und Archivierung von Komponenten
  - (nicht funktionale) Beschreibung der Komponenten (Qualität, potentielle Einsatzgebiete)
  - Restrukturierung, Aktualisierung bzw. Anhebung des Qualitätsniveaus der Komponenten, ggf. Auftrag zu *Development for Reuse*
  - Reorganisation der Bibliothek
- Kontinuierliche Information der Projektgruppen über verfügbare Komponenten
- Schulung von Projektmitgliedern in der Bibliotheksverwendung

Die Bedeutung des Software-Bibliothekars darf nicht unterschätzt werden. Es wäre empfehlenswert, wenn ein besonders fähiger Mitarbeiter diese Stelle übernimmt, der einerseits über umfangreiches Wissen über den Anwendungsbereich (*Domain-Wissen*) verfügt und andererseits auch besondere fachliche Qualitäten vorweisen kann.

c) Projektgruppen

Innerhalb der einzelnen Projektgruppen soll gezielt die Wiederverwendung von Software forciert werden. Dies kann zum einen durch eine spezielle Schulung geschehen, in die auch der Umgang mit der Software-Bibliothek einbezogen wird; insbesondere soll damit auch der Schwellenangst der Projektmitarbeiter im Zusammenhang mit der Bibliotheksverwendung entgegengewirkt werden; schließlich soll durch eine entsprechende Schulung das Vorurteil, daß der Suchprozeß und eine etwaige Abänderung einer gefundenen Komponente länger als das gänzlich neue Erstellen (vom Entwurf bis zum Test) dieses Bausteins dauert, entkräftet werden. Zum anderen können finanzielle Anreize eingesetzt werden, die sowohl das Schreiben von wiederverwendbarer Software als auch die Wiederverwendung von in der Software-Bibliothek abgelegten Komponenten prämieren. Es ist vor allem wesentlich, daß der Software-Entwickler nicht das Gefühl bekommt, an selbstproduzierten Mengen von Code-Zeilen gemessen zu werden.

Ein mögliches Modell einer entsprechenden Bewertungsstrategie besteht in der quantitativen Messung der Komponenten, die entweder vom jeweiligen Projektmitarbeiter aus der Software-Bibliothek wiederverwendet ("entliehen") oder von ihm erstellt und in die Bibliothek aufgenommen werden. Für die letztere Art von Komponenten wäre es denkbar, daß zusätzlich erhoben wird, wie oft diese wiederum zum Einbau in andere Systeme weiterverwendet werden. Aus diesen Parametern könnte nun der *Reuse-Bonus* berechnet werden.

d) Reuse-Entscheidungsteam

Zusätzlich zur Reuse-Gruppe, der die Administration von Reuse-Angelegenheiten obliegt, soll innerhalb der Organisation ein weiteres projektunabhängiges Team (*Reuse-Steering-Committee*) als oberster Reuse-Entscheidungsträger eingerichtet werden. Außer der Überwachung des Gesamtbestandes an Software-Komponenten soll dieses "Steering Board" sämtliche Software-Entwicklungs-Projekte im Hinblick auf Wiederverwendung überwachen und vor allem Konfliktsituationen zwischen den einzelnen Projekten und der (abgeschirmt) im Zentrum arbeitenden Reuse-Gruppe vermeiden bzw. lösen helfen.

Um dieses Ziel zu erreichen, wird insbesondere die Mitwirkung von Vertretern dieser Gruppe in Software Reviews hilfreich sein. Eine entscheidende Planungs- und Richtlinienkompetenz kommt diesem Reuse-Entscheidungsteam während der Einführungsphase zu. Weiters könnte es - in Fortführung dieser Aufgaben und in Berücksichtigung der relativ hohen Position dieser Gruppe im Unternehmen - auch zu ihrer Aufgabe gemacht werden, das Schulungs- und Technologie-Transfer Programm für Software-Entwickler auszuarbeiten und zu überwachen.

Selbstverständlich wird auch die kritische Beurteilung eines Reuse-Programmes gerade bei diesem - zwischen Tagesgeschäft und Bestandspflege liegendem - Gremium anzusiedeln sein. Um dies fair durchführen zu können, bedarf es freilich eines auf akkordierten Software-Metriken basierenden Reportings.

e) Reuse-Budget

Die Reuse-Gruppe soll über ein eigenes Budget verfügen können. Dieses Reuse-Budget soll im wesentlichen für adäquate Kosten-Nutzen-Analysen von Reuse, zur Erhebung und Bewertung des Reuse-Erfolges und zur Anschaffung von Reuse-Werkzeugen verwendet werden. Schließlich soll aus diesem Budget auch ein etwaiger Mehraufwand abgedeckt werden, der einzelnen Projekten

aus Reuse-Bestrebungen entsteht, damit das Konto des jeweiligen Projektmanagers in bezug auf zusätzliche Reuse-Kosten weitestgehend unbelastet bleibt.

Langfristig wäre denkbar und anzustreben, daß sich die Reuse-Gruppe ihr operatives Budget durch "Verkauf" von Komponenten an die Entwickler selbst "verdient". Da der Kostenträger evident ist, stellt dies keinerlei theoretische Probleme dar. Praktisch stellt sich dieser Selbstfinanzierung jedoch neben einer eklatanten Bepreisungs-Problematik allerdings die Motivations- und Lern-problematik entgegen. Aus diesen Gründen sind heute selbst noch "Reuse-Spitzenunternehmen" in der Phase des Nachdenkens über hausinterne Software-Preise. Die Reuse-Gruppe bleibt somit bis auf weiteres noch eine Gemeinkostenstelle des Entwicklers.

f) Kommunikationsstruktur

Wiederverwendung setzt Wissen über Wiederverwendungs-Potentiale voraus. Dieses ist nicht frei verfügbar, sondern kann nur durch aktive Kommunikationsmaßnahmen erreicht werden. Dabei ist es sowohl erforderlich, daß die Reuse-Gruppe über reuse-würdige Komponenten in laufenden Projekten informiert ist, wie auch daß die laufenden Projekte über den Inhalt der zentralen Bibliothek Bescheid wissen. Wie diese Informationen kommuniziert werden, und welcher Voll-ständigkeitsgrad dabei angestrebt werden sollte, hängt sehr stark von Inhalt und Umfang der Bibliothek ab.

Während obige Kommunikationsmuster jedoch "horizontal" zwischen Software-Entwicklern "gleicher Wellenlänge" ablaufen, setzt die Grundlage für ein erfolgreiches Wiederverwendungs-Konzept - wie wir in 2.1. gesehen haben - bereits bei Produkt- und Verkaufsentscheidungen an. Gerade aus dieser Sicht wird es insbesondere in jenen Unternehmungen, die Auftragsfertigung (a.) und (d.) durchführen, sinnvoll sein, vor Vertragsunterzeichnung unter Beiziehung von Meß- und Reuse-Experten einen Auftrag auf tatsächlich vorhandenes Reuse-Potential zu überprüfen. Zu groß ist sonst die Gefahr, daß "Details" übersehen werden, die aus einem "nahezu identischen Auftrag" eine fast komplette Neuentwicklung werden lassen.

2.4. Motivationsstruktur

Nicht nur fehlende Organisationsstrukturen und Prozeßmodelle, die speziell auf Reuse abzielen, sind dafür verantwortlich, daß Wiederverwendung von Software nicht schon längst gängige Praxis ist. Zusätzlich zu rein technischen Aspekten (z.B. fehlende Werkzeugunterstützung beim Suchen passender Software-Komponenten) spielen auch wirtschaftliche und psychologische Barrieren eine große Rolle, wenn es darum geht, die herkömmliche Strategie der Software-Entwicklung in Richtung Wiederverwendung zu verändern.

a) kostenrechnerische Aspekte

Unter Kostenaspekten muß vorerst klargestellt werden, daß Reuse doppelt kostet: sowohl bei der "Wiederverwendbarmachung" einer bereits vorhandenen Komponente, als auch bei deren Einbau in eine neu zu entwickelnde Problemlösung. Darüberhinaus entstehen noch Kosten der Erhaltung und Pflege der Reuse-Bibliothek (Reuse-Gruppe). Dennoch sollte die Summe dieser drei Kostenkomponenten deutlich geringer sein, als jene der wiederholten "Neu"-Entwicklung.

Aus Motivationssicht ist freilich festzustellen, daß die Ersparnisse nicht sofort eintreten und daß für die einzelnen beteiligten Projekte eingangs nur Kosten entstehen, denen kein (aus Sicht des Komponenten-Lieferanten) oder eingangs noch unwägbarer (aus Sicht des Komponenten-Konsumenten) Nutzen gegenübersteht.

Dies ist insbesondere aus Sicht des Lieferanten problematisch. Sicherlich ist es zweckmäßig, die für erfolgreiche Wiederverwendung erforderliche Anhebung des Abstraktions- und Qualitäts-

niveaus einer Komponente von den ursprünglichen Entwicklern durchführen zu lassen. Diesen fehlt aber meist nicht nur die erforderliche Zeit, sondern auch das nötige Budget. Um wenigstens eine dieser Hürden zu beseitigen, wird die Budgetierung eines Software-Hauses, das seine Bibliothek aus laufenden Projekten füllen möchte, eigene Ansätze für die Komponenten-Verbesserung zur Einbindung in das Archiv vorsehen müssen.

Daß der Betrieb der Software-Bibliothek in den ersten Jahren zu Lasten eines Zentral-Budgets erfolgen muß - aus dem auch die oben beschriebenen Adaptierungskosten zu decken sind - wurde bereits festgehalten. Verbleibt somit die Verwendung. Auch hier sind Entwickler wie Manager darauf hinzuweisen, daß Reuse nicht gratis ist. Statt einfach seiner Idee nachzugehen, muß der Entwickler nun in einem Archiv suchen, vielleicht einige Kandidaten genauer inspizieren, jedenfalls die als Zielkomponente erkannte noch (je nach Zertifikationsstand der Reuse-Bibliothek) austesten und gegebenenfalls adaptieren.

Insbesondere im Adaptierungsfall stellt sich die Frage, ob der von Einarbeitung bis Test reichende Adaptierungsaufwand die Reuse-Ersparnisse lohnt, weshalb viele Unternehmen auch lediglich "as-is reuse" favorisieren. Im allgemeinen sollte man die Konsultation des Archivs mit dem Besuch einer Bibliothek vergleichen können. Wenn innerhalb des Projekt-Zeitplanes auch eine solche (durchaus kurze) "Blätter- und Lesephase" vorgesehen wird, kann sich der einzelne nicht nur davon überzeugen, ob für das anstehende Problem nicht bereits eine Lösung existiert, sondern auch für den Fall, daß kein "as-is reuse" erzielbar ist, hilfreiche Anregungen für die "Neuentwicklung" holen.

b) psychologische Aspekte

Aus Sicht des *Software-Entwicklers* ist das sogenannte *Not-Invented-Here-Syndrom* wohl einer der bedeutendsten Gründe für die Ablehnung von Software Reuse. Darunter versteht man, daß Programmierer keine Komponenten, die sie nicht selbst entworfen und implementiert haben, in ihrer Arbeit verwenden wollen. Ein damit verbundenes Teilproblem ist die Angst um den Arbeitsplatz, da man sich selbst durch vorwiegende Verwendung von *fremdem* Code nicht mehr als unersetzbar fühlt. Außerdem würde es ein Anzeichen von Schwäche bedeuten, wenn man ein Programm nicht selbst schreiben kann. Ein anderer Gesichtspunkt in diesem Zusammenhang wird leicht übersehen, nämlich, daß es einfach mehr Spaß bereitet, ein Programm selbst zu schreiben. Hinzu kommt, daß viele Programmierer der Meinung sind, es sei leichter, ein Programm selbst zu schreiben, da es mit Schwierigkeiten verbunden sein kann, eine geeignete Komponente zur Wiederverwendung zu finden, ihre Funktionsweise zu verstehen und sie abzuändern, falls sie nicht zur Gänze unverändert einsetzbar ist.

Für weitere psychologische Aspekte, die mit Macht, Risiko und finanzielle Anreize des Programmierers als auch des Software-Managers verbunden sind, sei auf [Trac88], [Frak91] und [Walt91] verwiesen.

3. Umsetzungsaspekte

Ebensowenig, wie gesagt werden kann, daß Software Reuse einfach passiert, kann sich ein Unternehmen von heute auf morgen zu einer reuse-praktizierenden Organisation entwickeln. Die Einführung von Software-Wiederverwendung und alle damit verbundenen Restrukturierungsmaßnahmen können nur schrittweise durchgeführt werden und bedürfen kontinuierlicher Unterstützung aus der Führungsebene. So wird die Umstellung in einzelnen Phasen erfolgen müssen. Je nach Fortschritt werden unterschiedliche Maßnahmen in verschiedener Intensität erforderlich sein.

Sicherlich sind die in Kapitel 2 genannten Faktoren insgesamt wesentlich, um erfolgreich systematisch Software-Reuse zu betreiben. Während der Einführungsphase werden die Mitarbeiter jedoch sicherlich bereit sein, einen schrittweisen Wandel in Kauf zu nehmen, wenn entsprechende

sonstige Motivatoren gegeben werden. Folgende Voraussetzungen sollten jedoch bereits von Anbeginn gegeben sein bzw. so rasch wie möglich geschaffen werden.

Als in diesem Sinne grundlegende Faktoren können angesehen werden:

- *Verfügbarkeit von Wiederverwendbarem:* Dieser Aspekt wurde unter 2.3.a) behandelt. Unter Einsatz der auch dort schon angesprochenen Zertifizierungskennung wäre es denkbar, die Initialpopulation in der Bibliothek durch relativ freizügiges Einbringen von Komponenten aus Projekten rasch wachsen zu lassen. Im Zuge der Komponentenverwendung wird auch das Zertifikationsniveau angehoben und so höhere Sicherheit für Nachfrager erzielt.

- *"Reuse-Guru":* Die Existenz eines derartigen - auch mit ausreichender formaler Kompetenz ausgestatteten Sachpromotors ist immens wichtig, um gerade in der Einführungsphase Reuse-Barrieren zu überwinden. Von diesem "Reuse-Guru" wird auch starke Mitwirkung in anderen Bereichen des Technologie-Transfers und der technischen Weiterbildung zu fordern sein.

- *Musterprojekte:* Die Arbeit des "Reuse-Gurus" wird wesentlich erleichtert, wenn dieser auf Musterprojekte verweisen kann, in denen erfolgreiches Reuse betrieben wird oder wurde. Da sich jedoch Mißerfolg rascher herumspricht als Erfolg, sollte das Entstehen dieser Projekte nicht dem Zufall überlassen bleiben. Vielmehr wird man sowohl aus Sicht des Projektinhaltes (Reuse-Potential) als auch aus Sicht der personellen Besetzung, insbesondere der beiden oberen Führungseben innerhalb des Projektes (das kann mitunter das gesamte Projektteam sein) darauf achten, daß diese Mitarbeiter über hohe Motivation und Offenheit, aber auch technisches Knowhow verfügen.

- *"Experience dissemination & Technology transfer":* Guru und Musterprojekt können nur dann wirksam sein, wenn ihre Erfolgs-Stories auch unternehmensweit verbreitet werden können. Dazu ist allerdings auch wesentlich, daß die innerhalb von Mustgerprojekten erzielten Erfahrungen auch über ein entsprechend akzeptiertes System von Meßgrößen quantitativ nachvollziehbar sind. Freilich ersetzt dieser quantitative Beleg keineswegs die positive emotionale Grundeinstellung der Pioniere, denen auch die Chance geboten werden muß, diese zu verbreiten.

Unter Berücksichtigung obiger "Sperrhaken zum Erfolg" können folgende Phasen des organisatorischen Wandels zum reuse-orientierten Unternehmen angegeben werden:

- *Vorbereitungsphase:* In Abschnit 2.2.b) wurde festgestellt, daß erst auf der Stufe des "defined" Software Prozesses systematisches Reuse erzielbar ist. Da andererseits viele mitteleuropäische Software-Häuser noch unter der Schwelle zu dieser dritten Reifestufe stehen, wird in der Vorbereitungsphase alles zu unternehmen sein, um den (hier als existent angenommenen) Stufe-2 Prozeß (repeatable process) zu stabilisieren und allfällige Lücken insbesondere im Bereich der Qualitätssicherung und der Standards, wie auch in der Anwendung und Verbreitung (halb-)formaler Entwurfsverfahren und eines Meßsystems zu schließen.

  Weiters sind in der Vorbereitungsphase die Überlegungen für die Unterstützung auf motivationeller und organisatorischer Ebene zu treffen. Insbesondere werden die Schlüsselpersonen, die in der Einführungsphase Reuse unterstützen sollen, zu definieren und allfällige kostenrechnerische Veränderungen vorzunehmen sein.

- *Einführungsphase:* Mit ihrem Beginn muß sicherlich eine initiale Komponentenbibliothek vorliegen. Wenn diese, wie oben angeführt, initial noch relativ viel nicht-zertifizierte Komponenten enthalten sollte, wird das Reuse-Personal aufgerufen sein, möglichst rasch zertifizierte Komponenten anbieten zu können. Dies kann durch eigenes Komponenten-upgrade oder aber durch Verwendung und Rückmeldung aus Projekten (usage statistics, degree of modification statistics) erreicht werden. Weiters müssen nun die wichtigsten Rollen (Bibliothekar, Entscheidungsteam) mit Stelleninhabern besetzt sein.

Neben diesen Vorbereitungen sollte besonderes Augenmerk auf das Musterprojekt bzw. die Musterprojekte gelegt werden. Hierbei ist zu berücksichtigen, daß es eingangs noch zu Lerneffekten kommt, weshalb nicht sofort mit Kostensenkungen zu rechnen ist. Das Gegenteil wird kurzfristig der Fall sein. Ein "Sicherheits-Umweg" mag unter Umständen darin bestehen, als Alpha-Projekt für Wiederverwendung ein solches zu verwenden, in dem projekt-interne Wiederverwendung betrieben werden kann. Hierbei fallen viele Inhibitoren - und somit Fallstricke - weg. Die Gefahr liegt allerdings darin, daß wohl nur ein hinreichend großes Projekt über genügend Facettenreichtum verfügt, um aussagekräftig zu sein.

Allerdings kann erfolgreiches *intra-projekt-reuse* noch nicht als Abschluß der Einführungsphase angesehen werden. Sie findet ihren Abschluß durch erfolgreiches *inter-projekt-reuse* in ein oder zwei Musterprojekten im ursprünglichen Sinn.

- *Ausweitung:* Innerhalb dieser Phase soll das als erfolgreich erkannte Konzept der Wiederverwendung unternehmensweite Verbreitung erfahren. Um dies zu erreichen bedarf es vor allem entsprechenden Erfahrungsaustausches (dissemination) und Technologie-Transfers.

  Dazu ist festzuhalten, daß unter Erfahrungsaustausch keine Einbahnstraße verstanden werden darf. Unterschiedliche Bereiche eines Software-Hauses mögen unterschiedliche Rahmenbedingungen haben. Daher werden die aus den Musterprojekten gewonnenen Erkenntnisse zu überprüfen sein, und das Reuse-Konzept wird sich während seiner Verbreitung wandeln.

  Schließlich ist hier festzuhalten, daß die im Vorbereitungsteil angesprochenen Ausgangsvoraussetzungen nicht nur auf Unternehmensebene zu beachten sind. Sie gelten auch für die Projekt- und Mitarbeiterebene. Daher wird die Phase der Ausweitung auch durch entsprechende Forcierung der Mitarbeiterschulung charakterisiert sein.

  Schließlich soll noch der Aspekt der "*Reuse Kultur*" angesprochen werden. Es muß einfach schick sein, am Mittagstisch berichten zu können, welch tollen Wiederverwendungscoup man eben landete. Anderseits sollte das Eingeständnis, daß man eine Reuse-Möglichkeit verpaßt hat als mindestens so übel angesehen werden, wie wenn ein Review einen Initialisierungsfehler aufgedeckt hätte.

- *Erhaltung:* Da wir wissen, daß wir gute Vorsätze immer dann verbessen, wenn irgendwelche starke Motivatoren dagegen sprechen, ist es wichtig, Reuse-Orientierung nicht als Produkt zu betrachten, das fertig ist, wenn's jeder einmal gemacht hat, sondern als Prozeß, der laufend mit neuer Energie versorgt wird.

  Als wesentlichste Energieformen, die den Reuse-Prozeß am kochen halten, können
  - wahrnehmbare Pflege der Software-Bibliothek,
  - Marketing für den Inhalt der Software-Bibliothek,
  - Reuse-Forcierung in Reviews,
  - quantitative Beurteilung von Software-Projekten,
  - ordentliche und faire Projektaufwands-Schätzungen

  angesehen werden.

4. Zusammenfassung

In dieser Arbeit wurde gezeigt, daß die erfolgreiche Einführung einer Reuse-Strategie sich nicht nur auf software-technologische Aspekte beschränkt. Die Rahmenbedingungen für Erfolg- oder Enttäuschung eines Reuse-Vorstoßes werden lange bevor das erste Statement kodiert wird gelegt. So wie Industrialisierung in klassischen Produktionsbetrieben nicht einfach durch Mechanisierung handwerklicher Fertigung erzielbar war, gelingt dies auch im Software-Bereich nicht ohne generellem Umdenken auf allen Unternehmensebenen.

Weiters wurde dargelegt, daß - wie bei Industrialisierung - der zu wählende Reuse-Ansatz von der Produktpalette des Software-Hauses abhängt und daß auf einem noch unreifen Software-Prozeß kein erfolgreicher Reuse-Prozeß aufbauen kann.

Literatur

[Basi90] V.R. Basili: Viewing Maintenance as Reuse-Oriented Software Development", IEEE Software, Vol. 7, No. 1, Jan 1990, pp. 19-25.

[Basi91] V.R. Basili, G. Caldiera: "Identifying and Qualifying Reusable Software Components", IEEE Computer, Vol. 24, No.2, Feb. 1991, pp. 61-70

[Bisc89] W. Bischofberger, R. Keller: "Enhancing the Software Life Cycle by Prototyping", Structured Programming, No. 1, Springer Verlag New York Inc., 1989, pp. 47-59

[Boeh81] B.W. Boehm: "Software Engineering Economics", Prentice Hall, Englewood Cliffs, 1981

[Boeh88] B.W. Boehm: "A Spiral Model of Software Development and Enhancement", IEEE Computer, Vol. 21, No. 5, May 1988, pp. 61-72

[Börs91] J. Börstler: "Integrating Reuse into a Software Development Environment", in R.Prieto-Diaz, W. Schäfer, J. Cramer, S. Wolf (eds.): Proc. 1st International Workshop on Software Reusability, Dortmund, June 1991, pp. 234-238

[Dusi89] L. Dusink, P. Hall (eds.): "Software Re-use, Utrecht 1989", Workshop Overview and Conclusions, Proc. Workshops in Computing, Nov. 1989, pp. 7-20

[Floy84] Ch. Floyd: "A Systematic Look at Prototyping", in Budde et al. (eds.): "Approaches to Prototyping", Springer Verlag, 1984, pp. 1-18

[Frak91] B. Frakes: "A Survey of Software Reuse", in R.Prieto-Diaz, W. Schäfer, J. Cramer, S. Wolf (eds.): Proc. 1st International Workshop on Software Reusability, Dortmund, June 1991, pp. 65-70.

[Gall92] H. Gall, R. Klösch: "Reuse Engineering: Software Construction from Reusable Components", Proc. 16th Annual International Computer Software & Applications Conference, IEEE, Illinois, Sep. 1992, pp. 79-86

[Hend90] Henderson-Sellers B., Edwards J.M.: "The Object-Oriented Systems Life Cycle", Comm. ACM, Vol. 33, No. 9, Sept. 1990, pp. 142-159.

[Hoch92] E. Hochmüller: "AUGUSTA - Eine reuse-orientierte Software-Entwicklungsumgebung zur Erstellung von Ada-Applikationen", Dissertation, Universität Wien, Mai 1992

[Hump89] W.S. Humphrey: "Managing the Software Process", Addison Wesley, 1989.

[Mitt90] R.T. Mittermeir: "Software Wiederverwendbarkeit - Ein Ansatz zur Hebung von SW-Produktivität und SW-Qualität", Proc. 9. Internationaler Kongress Datenverarbeitung im Europäischen Raum "EDV in den 90er Jahren: Jahrzehnt der Anwender - Jahrzehnt der Integration", ADV, Wien, 1990, pp. 158-167.

[Mitt91] R.T. Mittermeir: "POWDER - A Recursive Methodology for Prototyping of Wicked Development Efforts with Reuse"; Interner Bericht 4/91, Klagenfurt, April 1991

[Mitt92] R.T. Mittermeir u. R.A. Schlemmer: "Stepwise Improvement of the Software-Process in a Multidimensional Framework", in: Annual Review of Automatic Programming, Vol. 16/2, Pergamon, 1992, pp. 63-70

[Oste87] L. Osterweil: "Software Processes are Software Too", Proc. 9th Intl. Conference on Software Engineering, IEEE, Monterey, 1987, pp. 2-13

[Prie91a] R. Prieto-Diaz, G. Arango (eds.): "Domain Analysis and Software Systems Modeling", IEEE-CSP Tutorial, Los Alamitos, 1991

[Prie91b] R. Prieto-Diaz: "Making Software Reuse Work: An Implementation Model", in R.Prieto-Diaz, W. Schäfer, J. Cramer, S. Wolf (eds.): Proc. 1st International Workshop on Software Reusability, Dortmund, June 1991, pp. 86-92

[Trac88] W. Tracz: "Software Reuse: Motivators and Inhibitors", in W. Tracz (ed.): IEEE Tutorial "Software Reuse: Emerging Technology", 1988, pp. 62-67

[Trac90] W. Tracz: "Where Does Reuse Start?", ACM SIGSOFT Software Engineering Notes, Vol. 15, No. 2, Apr. 1990, pp. 42-46

[Walt91] P. Walton: "Software Reuse: Management Issues", in R.Prieto-Diaz, W. Schäfer, J. Cramer, S. Wolf (eds.): Proc. 1st International Workshop on Software Reusability, Dortmund, June 1991, pp. 100-106.

[Yeh 91] R. Yeh, D. Naumann, R. Mittermeir, R. Schlemmer, W. Gilmore, G. Sumrall, J. LeBaron: "A Commonsense Management Model", IEEE Software, Vol. 8, No. 6, Nov. 1991, pp. 23-33.

Biographie

Elke Hochmüller ist Universitätsassistentin am Institut für Informatik der Universität Klagenfurt. Ihre Forschungsschwerpunkte liegen im Bereich Software Engineering, insbesondere sind dies Software Reuse, Software Engineering Environments, Software-Prozeßmodelle und (objektorientierter) Software-Entwurf.

Roland Mittermeir ist Professor für Informatik mit besonderer Berücksichtigung der betrieblichen Anwendung an der Universität Klagenfurt. Er beschäftigt sich in Forschung und Lehre schwerpunktmäßig mit Software Engineering, wobei der Bereich der systematischen Software-Wiederverwendung und die Gestaltung des Software-Entwicklungsprozesses eine besondere Stellung einnehmen.