Software-Wiederverwendung

Theoretische Grundlagen, Vorteile und realistische Beurteilung

Arif Chughtai und Oliver Vogel
mail@arifchughtai.org, mail@ovogel.de
Version 1.0 30. Juli 2001

Abstrakt

Aktuell ist Software-Entwicklung oft sehr ressourcen-intensiv und von Misserfolg gekrönt. Ein grosse Chance, diesen Zustand zu verbessern, bietet hier die Software-Wiederverwendung.

Bei der Einführung von Software-Wiederverwendung in einem Unternehmen genügt es jedoch nicht, einfach Code wiederzuverwenden. Vielmehr muss ein systematischer, ganzheitlicher, von allen Unternehmensteilen voll unterstützter Ansatz zum Zuge kommen. Dabei spielen dann Reuse-Assets (Dokumentation, Architekturen, Patterns, Code etc.), Prozesse, kulturelle und soziale Faktoren eine Rolle.

Für IT-Unternehmen wird der erfolgreiche Einsatz von Software-Wiederverwendung in Zukunft einer der entscheidenden Faktoren im Sein- oder Nicht-Sein im Wettbewerb bedeuten.

Dieser Artikel beleuchtet die angesprochenen Themenkreise und gibt Antworten auf folgende Fragen:

- Was müssen IT-Unternehmen beachten, wenn sie Software-Wiederverwendung einführen möchten?
- Wo liegen die Hindernisse?
- Haben IT-Unternehmen einen echten Nutzen von Software-Wiederverwendung?
- Und vor allem, was ist Software-Wiederverwendung überhaupt?

Inhaltsverzeichnis

1. Einleitung

1.1 Motivation

Die inhärente Komplexität von Software-Systemen
Die Notwendigkeit von Software-Wiederverwendung ergibt sich aus der inhärenten Komplexität von Software-Systemen. Software-Systeme sind oftmals so umfangreich, dass die Kapazität der menschlichen Intelligenz nicht ausreicht, um sie komplett zu erfassen. Wenn es Software-Entwicklern nicht gelingt, mit dieser Komplexität umzugehen, enden Projekte mit Zeit- und Budgetüberschreitungen und erfüllen nicht alle Anforderungen der Auftraggeber ([1], S. 23).

Konsequenzen der inhärenten Komplexität von Software-Systemen [2]
- Fünf von sechs Software-Projekten sind nicht erfolgreich
- Ein Drittel aller Software-Projekte werden gestoppt
- Die restlichen Projekte benötigen doppelt soviel Zeit und Geld wie ursprünglich geplant
- Software-Systeme sind nicht wart- oder erweiterbar
- Software-Systeme erfüllen nicht die Anforderungen [2]

Bis heute hat sich die Wiederverwendung von Software als eine der wichtigsten Techniken zur Bewältigung der inhärenten Komplexität herausgestellt ([3], S. 1 [17], S. 7 [20], S. 1).

Brian Morrow, Direktor Component Based Development (CBD) von Texas Instruments geht sogar davon aus, daß der Einsatz von Software-Wiederverwendung über das Überleben eines Unternehmens entscheiden wird ([27], S. 2).

Organisatorische und technologische Faktoren als Motivator für Software-Wiederverwendung
Viele IT-Organisationen arbeiten projektorientiert, d.h. zur Erbringung ihrer Dienstleistungen werden Mitarbeiter für die Dauer eines Projektes organisationsübergreifend zu Teams zusammengefaßt. Daraus ergibt sich die für das Projektmanagement typische Matrix zwischen Unternehmens- und temporärer Projektorganisiation. Obwohl hierdurch eine Verzahnung zwischen einzelnen Fachabteilungen, wie z.B. objektorientierte Technologien, Datenmodellierung oder Middleware und den verschiedenen Projekten erreicht wird, handelt es sich bei einem Grossteil der Projekte um eigene kleine Inseln, die vom Rest des Unternehmens entkoppelt sind. Wozu dies führen kann, wird im folgenden durch ein kleines Bild verdeutlicht:

Ein Unternehmen kann man sich als ein Meer vorstellen, in dem sich verschiedene Inseln (Projekte) befinden. Zwischen den Inseln gibt es keinen formellen Informationsaustausch, weder mittels Flaschenpost, Rauchzeichen oder Postschiffen. Informationen fließen meistens nur, wenn jemand von einer Insel zur anderen übersetzt oder sich Bewohner unterschiedlicher Inseln auf dem Meer treffen. Durch diese informellen Brücken werden dann Informationen informell von einer Insel zur anderen transportiert. Wenn nun die Bewohner einer Insel z.B. eine Getreidemühle bauen möchten, ist vielleicht jemand unter ihnen der bereits auf einer anderen Insel an dem Bau einer Getreidemühle beteiligt war oder der einen kennt, der eine Getreidemühle gebaut hat. Wenn beide Fälle nicht zutreffen, beginnen die Inselbewohner beim Bau der Getreidemühle ganz von vorne, wenngleich irgendjemand auf einer anderen Insel bestimmt schon einmal eine Getreidemühle gebaut hat.

Diese Metapher zeigt sehr gut, daß neben der Projektorganisation auch gelebte Informationsprozesse und -instrumente vorhanden sein müssen, um den Erfolg der

Projekte und damit den Erfolg der Organisation zu garantieren. Dem durch die Metapher veranschaulichten Problem stehen sehr viele IT-Organisationen gegenüber. Nahezu 75% aller Software-Entwicklungs-Unternehmen befinden sich auf Level 1 des Software Process Capability Maturity Models (SPCMM), das von dem Software Engineering Institute (SEI) der Carnegie Mellon Universität entwickelt wurde ([13], S. 1). Auf Level 1 gibt es nur einen adhoc-Software-Prozeß und wenige definierte Prozesse. Darüber hinaus ist Level 1 durch chaotische Vorgehensweisen gekennzeichnet.

Sofware-Wiederverwendung als Mittel für Knowledge-Management
Wie können IT-Organisationen diesem Problem begegnen? Zu aller erst muss eine Basis für eine projektübergreifende Kommunikation geschaffen werden. Dies kann z.B. durch die Etablierung von Communities in denen sich Interessierte zu bestimmten Themen, wie z.B. Design Patterns, XML, Java, Security, treffen und austauschen können, erreicht werden. Dies sind Plattformen auf denen ein Wissenstransfer und somit letztlich auch die Wiederverwendung von Wissen stattfinden kann. Der Aktualisierung und dem Erwerb von Wissen kann durch gezielte interne Ausbildungsprogramme erreicht werden. Einige IT-Organisationen errichten zu diesem Zweck spezielle Akademien. Diese Akademien begleiten jeden Mitarbeiter während seiner Laufbahn und stellen an seine Bedürfnisse anpassbare Aus- und Weiterbildungspläne bereit. Neben dem Austausch und dem Erwerb von Wissen mittels Communities und Akademien ist gerade die Wiederverwendung von Software von essentieller Bedeutung, da Wissen auch in Software transformiert wird. Knowledge-Management in einem IT-Unternehmen ist also ohne Software-Wiederverwendung nur ein inkonsequentes Vorgehen.

Der Wiederverwendungskontext
Als eine der technischen Grundvoraussetzungen für Software-Wiederverwendung muss es eine Möglichkeit geben, sich einen Überblick und Zugriff zu verschaffen über und auf Software-Artifakte in einem Unternehmen. Dies wird idealerweise mit Hilfe eines Repositories dieser Software-Artifakte realisiert.

Bei Vorüberlegungen zu einem solchen Repository zeigt sich aber schnell, dass es schwierig ist, dass Repository isoliert zu betrachten. Vielmehr muss der ganze Wiederverwendungskontext (kulturelle- und soziale Aspekte, Prozesse, Wettbewerbsvorteile etc.) behandelt werden und dabei wäre dann das Repository nur ein Baustein.

Damit ergibt sich die Situation, daß der erste Schritt in der Etablierung von Software-Wiederverwendung nicht ein Repository sein kann, sondern dass der gesamte Kontext von Software-Wiederverwendung berücksichtigt werden muss.

1.2 Ziele und Aufbau

Das Dokument richtet sich an Management und Verantwortliche von technischer Seite.
Hauptziele dieses Dokuments sind es, einen Überblick zu geben über Aspekte und den Prozess bei der Einführung von Software-Wiederverwendung in einem Unternehmen sowie bewusst zu machen, dass Software-Wiederverwendung nicht bloss ein Schlagwort ist, sondern über die zukünftige Überlebensfähigkeit eines IT-Unternehmens entscheiden wird.

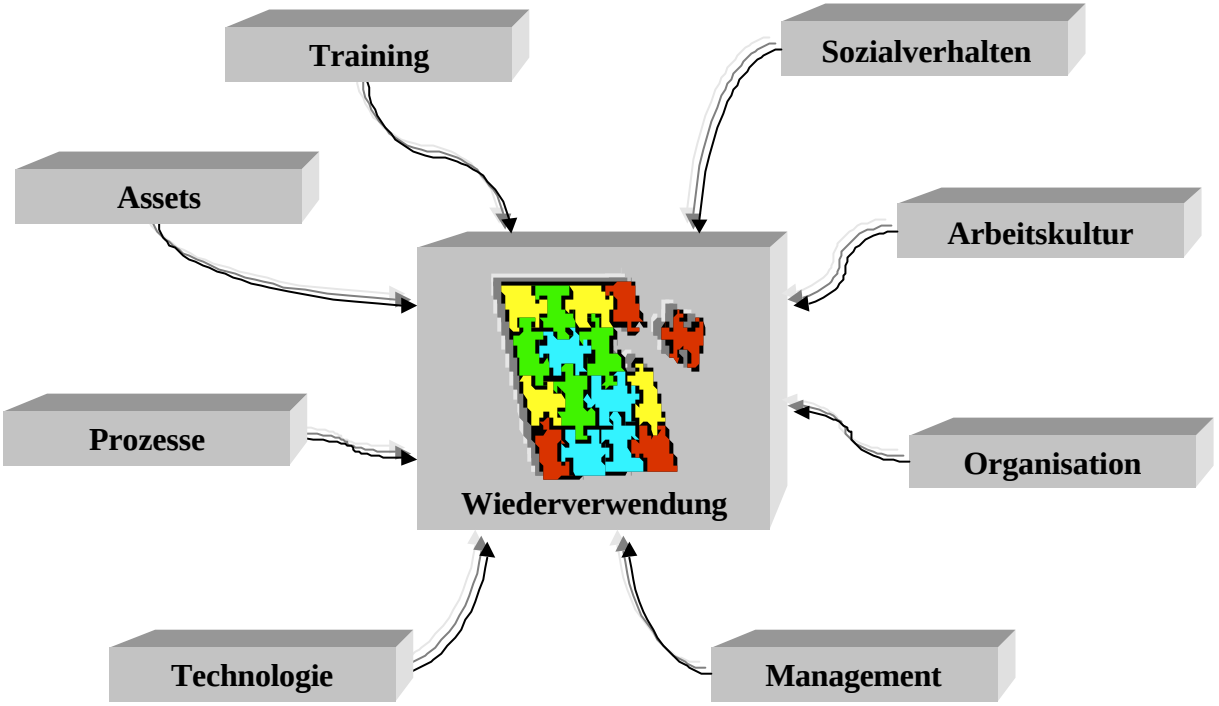
Kapitel 2 zeigt Aspekte bei der Einführung von Software-Wiederverwendung in einem Unternehmen.
Kapitel 3 beschäftigt sich mit der Situation in Unternehmen unter Software-Wiederverwendungs-Gesichtspunkten.
Kapitel 4 beurteilt abschließend die in Kapitel 3 dargestellten Erkenntnisse, indem dabei die Relevanz von Software-Wiederwendung in der heutigen Zeit berücksichtigt wird. Des weiteren werden Zukunftsperspektiven aufgezeigt.

2 Software-Wiederverwendung als Lösungsmittel

2.1 Überleitung

In diesem Kapitel wird Software-Wiederverwendung als ein Weg gezeigt, der die oben gezeigten Probleme der traditionellen Software-Entwicklung zu vermeiden hilft. Zunächst werden wichtige konzeptionelle und technologische Begriffe im Zusammenhang mit der Software-Wiederverwendung erläutert. Anschliessend wird das Vorgehen bei der Realisierung von Software-Wiederverwendung behandelt.

In folgender Grafik sind die wichtigsten, im weiteren näher erläuterten Faktoren für erfolgreiche Software-Wiederverwendung im Überblick dargestellt:



2.2 Konzeptionelle und technologische Mittel

Es folgt eine Erläuterung der wichtigsten konzeptionellen und technologischen Möglichkeiten um Software-Wiederverwendung zu erreichen. Später wird erläutert, daß diese Mittel für sich allein stehend keine echte Software-Wiederverwendung bewirken können. Auch wird deutlich, daß Software-Wiederverwendung sich nicht auf die Wiederverwendung von Code beschränkt, sondern alle Artifakte (Architektur, Design etc.) entlang des Entwicklungszyklus eines Software-Systems involviert.

2.2.1 Analysis-Patterns

Analysis-Patterns [7] zeigen Lösungen für bestimmte Aspekte bei der Erstellung von Analyse-Modellen vertikaler Problembereiche wie Gesundheits- oder Rechnungswesen.

Als Beispiel sei hier ein Pattern für die Problematik des Messens genannt ([7], S. 45f), welches einen Lösungsweg aufzeigt, Messungen (im konkreten Fall Messungen an einem Patienten) zu modellieren.

2.2.2 Architektur-Patterns

Eine Software-Architektur ([4], S. 12) beschreibt die grundlegende Strukturierung eines Software-Systems in Subsysteme, welche Verantwortlichkeiten diese Subsysteme wahrnehmen und wie diese miteinander kollaborieren. Architektur-Patterns liefern

Architektur-Schablonen für verschiedene häufig vorkommende Ausrichtungen (Benutzerinteraktion, Client-Server, verteilt etc.) von Software-Systemen.

Das Model-View-Controller ([4], S. 125f) Pattern ist ein prominentes Beispiel für ein Architektur-Pattern. Es ermöglicht ein und dasselbe Modell (Anwendungsfunktionalität) unterschiedlich darstellen zu können. Damit ist es z.B. möglich das gleiche Modell auf verschiedenen Betriebssystemplattformen visuell darstellen zu können.

Einzelne Architektur-Patterns behandeln immer nur bestimmte Teilaspekte der Architektur eines Software-Systems. Um die Gesamtarchitektur eines Software-Systems zu beschreiben ist es notwendig,verschiedene Architektur-Patterns einzusetzen. Hierdurch lassen sich dann auch generische Gesamtarchitekturen (Architektur-Blueprints) für immer wieder benötigte Zielarchitekturen entwickeln.

Ein konkretes Beispiel für eine generische Gesamtarchitektur ist SUN`s J2EE [25].

2.2.3 Design-Patterns

Design-Patterns ([9], S. 2f) sind Design-Schablonen für immer wiederkehrende Problemstellungen beim feingranularen Design eines Software-Systems. Als Beispiel wird hier das Adapter-Pattern ([9], S. 139f) erwähnt welches das Lösungsdesign liefert für die Anforderung, dass verschiedene Funktionen mit identischem Verhalten benutzt werden können, ohne für jede dieser Funktionen die aufrufende Funktion anpassen zu müssen.

2.2.4 Klassenbibliotheken

Klassenbibliotheken stellen eine Sammlung an unterschiedlicher, auf Klassen verteilte, Funktionalität bereit, welche im Gegensatz zu der Funktionalität in Frameworks kaum “verdrahtet” ist und vom Entwickler aufgerufen werden muss.

Die C++ Standard-Template-Library wäre ein Beispiel für eine Klassenbibliothek.

2.2.5 Frameworks

Frameworks [23] sind Software-Infrastrukturen für ganze Problembereiche (Domänen), z.B. Finanz- oder Gesundheitswesen. Dabei muss sich der Entwickler nur noch um die Anwendungsteile kümmern, welche vom Framework nicht abgedeckt werden können, z.B. kundenspezifische Anforderungen. Ein wichtiges Merkmal von Frameworks ist die Hollywood-Metapher “Don´t call us. We call you”. Darin kommt zum Ausdruck, dass der Entwickler überwiegend nicht Funktionalität des Frameworks aufruft, sondern dass das Framework Funktionaliät aufruft, welche der Entwickler gemäss den Konventionen des Frameworks bereitstellt.

Implementierungen von CORBA [22] sind Beispiele für technische Frameworks welche Infrastrukturen für verteilte Objekte bereitstellen.

2.2.6 Komponenten

Definition [5]:

Eine Komponente ist ein Stück Software mit einer definierten Schnittstelle. Desweiteren deckt eine solche Komponente genau eine Domäne ab und ist stark kontextabhängig. Komponenten können dazu genutzt werden zu einer neuen Komponente zusammengebaut zu werden.

Komponenten sind intern aus Frameworks aufgebaut und interagieren mit Hilfe von Frameworks.

Enterprise-Java-Beans [18] und COM+ [22] sind Beispiele für Komponentenmodelle in der Praxis.

2.2.7 Software-Assets

Software-Wiederverwendung geschieht, indem bei der Software-Entwicklung auf sogenannte Software-Assets ([6], S. 1) (im Folgenden als Assets bezeichnet) zurückgegriffen wird. Diese Assets umfassen nicht nur den Code, sondern alles, was in Zusammenhang mit diesem steht; Domänenanalyse, Anforderungsbeschreibung, Spezifikation, Architektur, Design, Frameworks, Componenten, etc.

Erfolgreiche Software-Wiederverwendung kann sich nicht ausschliesslich auf Code stützen, sondern muß den gesamten Kontext (sämtliche Dokumentation, Architektur und Design, d.h. die Assets, in deren Kontext ein bestimmter Code steht) eines bestimmten Codes berücksichtigen. Nur so ist gewährleistet, daß bestimmter Code auf seine Zweckmässigkeit für eine konkrete Problemstellung beurteilt werden kann und auch „richtig“ verstanden wurde und damit auch in beabsichtigter Weise benutzt (wiederverwendet) wird.

2.2.8 Asset-Repositories

Asset-Repositories ([29], S. 4) dienen dem Zugriff auf Assets. In der Literatur stösst man oft auf den Begriff Component-Repository. Dieser Begriff ist jedoch zu code-lastig. Asset-Repositories implizieren Component-Repositories.

Innerhalb von IT-Organisationen entstehen in verschiedenen Projekten Assets von denen andere Projekte nichts erfahren. Z.B. ist das Thema Object-Relational-Mapping in sehr vielen Projekten von äußerster Wichtigkeit. Trotzdem gibt es zu diesem Thema oft keine unternehmensweiten Basis-Lösungen. Auf Grund dessen muß das Rad oftmals neu erfunden werden. Eine Konsolidierung dieser Assets würde diese Tatsache verhindern. Indem in einem Asset-Repository alle Assets gehalten und jedem Projekt zur Verfügung gestellt werden, kann der oben erwähnte Mehrwert erzielt werden. Es ist also äußerst wichtig, ein Asset-Repository zu erschaffen, da dieses als Kommunikationsmedium über Projekte hinweg genutzt werden könnte.

2.2.9 Wiederverwendungsebenen

Folgende Tabelle gibt einen Überblick darüber in welchen Bereichen (Wiederverwendungsebenen) die einzelnen konzeptionellen und technologischen Mittel Wiederverwendung bewirken:

	Wiederverwendungsebene			
	Analyse	Architektur	Design	Code
Analysis-Patterns	x			
Architektur-Patterns	x	x		
Design-Patterns			x	
Klassenbibliotheken			x	x
Frameworks	x	x	x	x
Komponenten	x	x	x	x

Tabelle 1 : Wiederverwendungsebenen konzeptioneller und technologischer Mittel

Die Architektur eines Software-Systems wird auch von den Ergebnissen aus der Analyse geprägt. Aus diesem Grund erlauben Architektur-Patterns neben der Wiederverwendung der Architekur auch Wiederverwendung von Ergebnissen aus der Analyse bei Software-Systemen des gleichen Problembereichs. Weil sich die Architektur auf die grobgranularen Aspekte eines Software-Systems fokusiert bewirkt sie keine Wiederverwendung von Design und Code.

Klassenbibliotheken haben ihren Schwerpunkt bei der Wiederverwendung von Code. Klassen wie String, List etc. bringen für Analyse oder Architektur keinen und nur wenig Mehrwert für das Design eines Software-Systems.

Sowohl Frameworks wie auch Komponenten sind die Mittel mit dem grössten Wiederverwendungsfaktor. Beide haben den Anspruch neben Code auch Ergebnisse aus Analyse, Architektur und Design für ganze Problemberreiche in sich zu tragen.

2.3 Hindernisse

Ganz wesentlich für den Erfolg von Software-Wiederverwendung in einem Unternehmen sind kulturelle und soziale Faktoren (mehr noch als die technisch-organisatorischen). Der Zustand dieser beiden Faktoren in IT-Unternehmen ist oft alles andere als der Software-Wiederverwendung dienlich.

In diesem Zusammenhang können folgende in Relation zueinander stehende Punkte genannt werden ([6], S. 9 [15], S. 33 [27], S. 1 [28], S. 2f):

- **Fehlende Kommunikation zwischen Projekten / Fehlender Know-How-Transfer im Unternehmen**. Die Hauptgründe liegen hier oft darin das zum einen keine kommunikationfördernden Strukturen und Prozesse existieren und das zum anderen zwischen verschiedenen Projekten eine Art von Konkurrenz gepaart mit wechselseitiger Geringschätzung besteht

- **Mitarbeiter sind nicht bereit ihre Arbeitsergebnisse detailiert einer breiten Öffentlichkeit preiszugeben**; z.B. aus Angst, dass Fehler entdeckt werden, um die eigene Stellung zu sichern etc.

- **Fehlende Bereitschaft etwas (z.B. Software) zu benutzen, daß woanders entwickelt worden ist** (Not Invented Here Syndrome). Dies kann daran liegen, daß sich Mitarbeiter bzw. Projekte profilieren wollen indem sie etwas from-scratch neu entwickeln oder daß diese kein Vertrauen (z.B. Aufgrund von Geringschätzung, ungenügender Kommunikation etc.) in die Qualität woanders entwickelter Software haben

Folgende technisch-organisatorischen Faktoren können dem Wechsel eines Unternehmens hin zu Prinzipien der Software-Wiederverwendung massiv im Weg stehen und sollten bei der Einführung von Software-Wiederverwendung vorrangig angegangen werden ([6], S. 9, 15):

- **Fehlendes Commitment des Managements zu Software-Wiederverwendung**. Dies führt dazu das Software-Wiederverwendung nicht systematisch betrieben werden kann weil hierfür finanzielle und organisatorische Anstrengungen notwendig sind, die vom Management lanciert werden müssen

- **Software-Wiederverwendung wird nicht belohnt,** sondern eher bestraft, weil Software-Wiederverwendung zu beginn immer ressourcenintensiver ist. Erfolg wird oft eher an den Lines-Of-Code anstatt an der Qualität der gelieferten Software gemessen. Dadurch wird die Motivation der Mitarbeiter nicht gesteigert, da sie für

eine Partizipation (Entwicklung von Assets und deren Verwendung) an der Software-Wiederverwendung nicht belohnt werden.

- **Es exisitiert kein Überblick zu den Assets einzelner Projekte**. Damit ist es nicht ohne weiteres möglich, zu sehen, ob etwas in einem anderen Projekt bereits vorhanden ist und genutzt werden kann

- **Es werden zu wenig Ressourcen bereitgestellt**. Es fehlt die Erkenntniss, daß Software-Wiederverwendung nicht einfach von selber läuft, sondern "betrieben" werden muß. Hierfür ist es jedoch unabdingbar in ausreichendem Umfang Mitarbeiter und finanzielle Mittel zur Verfügung zu haben

- **Fehlender systematischer Ansatz** für die Einführung von Software-Wiederverwendung in einem Unternehmen. Es wird versucht ad-hoc mit lokalem Fokus (ohne das die Unternehmensleitung involviert ist bzw. ihr Commitment gegeben hat) in Abteilungen, Projekten oder Teams Software-Wiederverwendung zu betreiben und nur der Code anstelle von Assets als relevant erachtet wird

- **Handeln ist auf kurzfristigen Erfolg ausgelegt**. Unter strategischen Gesichtspunkten ist Software-Wiederverwendung als eine Maßnahme zu sehen, die mittel- und langfristig den Erfolg steigern wird, d.h. wenn, wie so oft, nur der kurzfristige Erfolg im Vordergrund steht wird Software-Wiederverwendung keine bedeutende Rolle spielen

- **Projekte bzw. Mitarbeiter führen ein Inseldasein**. Darin kommt zum Ausdruck, daß keine Kommunikation stattfindet. Kommunikation ist jedoch einer der Grundpfeiler für erfolgreiche Software-Wiederverwendung

- **Projektplanung, die zu Zeitdruck führt**. Unter Zeitdruck gehören Aspekte der Software-Wiederverwendung zu den ersten Kandidaten, die über Bord geworfen werden

- **Software-Wiederverwendung wird nur halbherzig durchgeführt**. Dies ist meist darauf zurückzuführen, daß Mitarbeiter nicht ausreichend überzeugt worden sind vom Nutzen der Software-Wiederverwendung oder das ein systematischer Ansatz fehlt und dadurch Software-Wiederverwendung eher zu einer Belastung für die Mitarbeiter wird und deshalb nach Möglichkeit vermieden wird

- **Unzureichende Einführung und Dokumentation von Assets**. Software, welche

nicht verstanden worden ist, hat schlechte Karten benutzt zu werden

2.4 Umsetzung

2.4.1 Vorabinvestitionen

Bevor ein Unternehmen auf Software-Wiederverwendung setzen kann, sind zunächst Vorabinvestitionen zu tätigen. Die Entwicklung einer wiederverwendbaren Komponente kostet zwischen dem 1.0 und 2.5 fachen der Entwicklung von traditioneller Software ([14], S. 385). Extrapoliert sind dies auch die Kosten für die Reuse-Organisation, da die Organisation die Komponente baut.

Dabei wird nicht ein einzelnes Projekt die Last tragen können, die notwendig ist, um die Grundlagen zu legen für die unternehmensweite Einführung von Software-Wiederverwendung. Vielmehr müssen die Aufwände auf verschiedene Projekte (Referenzprojekte) verteilt und die Projekte auch subventioniert werden ([6], S. 7).

Für IT-Unternehmen, die ihren Schwerpunkt in Entwicklung und Dienstleistung haben, ergeben sich folgende Felder mit erhöhtem Investitionsaufwand ([6], S. 6):

- Domänenanalyse
- Entwicklung
- Integration
- Testen
- Quality-Assurance
- Wartung
- Mitarbeiter
- Evaluierung und Einführung von Tools

Als Tools, welche für Software-Wiederverwendung von besonderer Bedeutung sind, kann man sicherlich Asset-Repositories ([29], S. 3f) mit umfangreicher Suchfunktionalität nennen. denn eine der Grundvoraussetzungen für Software-Wiederverwendung ist es das ein Entwickler sich einen Überblick über die Assets im Unternehmen verschaffen und auf diese zur Nutzung auch entsprechend komfortabel und effizient zugreifen kann.

Amortisieren werden sich diese Investitionen, indem die Ergebnisse der Pilot-Projekte im Laufe der Zeit in vielen weiteren Projekten Nutzniesser finden (siehe Kapitel 2.5).

2.4.2 Durchführung

Um Software-Wiederverwendung in einem Unternehmen erfolgreich einzuführen und am Leben halten zu können, müssen folgende Voraussetzungen geschaffen werden ([6], S. 6 u. 16 [27], S. 1 [28], S. 2 [29], S. 4):

- **Eine unternehmensweite Sensibilität ist ein unbedingtes Muss** für den Erfolg von Software-Wiederverwendung. Deshalb muß an erster Stelle das Commitment des Managements zur Etablierung einer Software-Wiederverwendungs-Kultur stehen. Erst wenn dieses Fundament geschaffen wurde, können die weiteren Voraussetzungen für die Einführung von Software-Wiederverwendung geschaffen und vorhandene Hindernisse entfernt werden.

- **Anpassung der organisatorischen Strukturen**. Es sind Mitarbeiter vorzusehen, welche sich vorrangig mit Aufgaben rund um die Software-Wiederverwendung beschäftigen. Bei diesen Aufgaben handelt es sich um Training, Tool-Wartung, Management von Assets

- **Bereitstellung von Tools**, welche die Software-Wiederverwendung unterstützen (z.B. ein Asset-Repository). Software-Wiederverwendung darf nicht zu ständig erhöhtem Aufwand für die Mitarbeiter führen, sondern sollte, wo immer möglich, automatisiert werden

- **Definition von Prozessen für Software-Wiederverwendung**. Neue Mitarbeiter müssen für Software-Wiederverwendung sensibilisiert werden. Projekte müssen derart aufgesetzt und durchgeführt werden, dass sie Software-Wiederverwendung berücksichtigen, d.h. insbesondere das verhindert wird, dass in einem Projekt das Rad neu erfunden wird. Es muss festgelegt werden, wie Assets in ein Asset-Repository einzubringen und wie diese zu benutzen sind. Es ist zu definieren, wie Assets zu pflegen (Bugfixing, Anpassung) sind. Art und Weise regelmässiger Kommunikation zwischen verschiedenen Projekten und Teams sind festzulegen

- **Schaffung einer Arbeitskultur, welche der Software-Wiederverwendung entgegenkommt**. Diese Arbeitskultur muss geprägt sein von offener team- und projektübergreifender Kommunikation sowie der Bereitschaft Assets bereitzustellen und auch zu nutzen

- **Fortlaufende Ausbildung** ist eine Grundvorausetzung für eine erfolgreiche Software-Wiederverwendung. Assets müssen eingeführt und geschult werden. Ebenso die verwendeten Tools und Technologien. Darüberhinaus ist Training für notwendige Soft-Skills (z.B. Kommunikationsfähigkeit) vorzusehen

- **Fortlaufender Wissenstransfer** zwischen Mitarbeitern, Projekten und Teams. Hier ist das Thema Kommunikation adressiert. Es kann nur etwas wiederverwendet werden, was auch bekannt ist. Dazu muß es ersteinmal bekanntgemacht, d.h. kommuniziert werden. Dies kann und sollte über verschiedene Kanäle z.B. Communities erfolgen

Ziel muss es sein, Software-Wiederverwendung in einem Unternehmen derart zu etablieren werden, dass diese auch in schwierigen Projektphasen nie zur Disposition steht - Software-Wiederverwendung ist kein nice to have, sondern ein must have.

Bei der Umsetzung von Software-Wiederverwendung in einem Unternehmen ist eine ganze Software-Wiederverwendungsinfrastruktur zu schaffen. Diese Infrastruktur umfasst Management für die Software-Wiederverwendung, Prozesse, Tools, Training/Wissenstransfer für Mitarbeiter/Projekte.

Es folgt eine unvollständige „To Do“-Liste zu den einzelnen notwendigen Aspekten bei der Durchführung([13], S. 1 [15], S. 17f [28], S. 3f [29], S. 4):

Es sind Prozesse zu definieren für
- Bereitstellung von Assets
- Einführung von Assets
- Einführung von Software-Wiederverwendung bei Mitarbeitern und Projekten
- Wartung von Assets
- Wissenstransfer und Kommunikation zwischen Mitarbeitern, Teams und Projekten

Es sind Mitarbeiter-Ressourcen zu schaffen für
- Aufbau und Wartung von Assets
- Identifikation von Assets
- Kommunikationsschnittstellen zwischen Projekten

- Quality-Assurance
- Training
- Unterhalt eines Asset-Repositories

Es ist Training vorzusehen für
- Assets
- Prinzipien der Software-Wiederverwendung
- Tools

Es sind Management-Aktivitäten notwendig für
- Belohnung für Bereitstellung und Nutzung von Assets
- Unternehmensweites Commitment zu Software-Wiederverwendung
- Identifikation geeigneter Projekte für Software-Wiederverwendung
- Projektplanung, die Software-Wiederverwendung berücksichtigt
- Unterstützung von Verhaltensweisen, welche für Software-Wiederverwendung förderlich sind

Die vorgestellten Elemente der Software-Wiederverwendungsinfrastruktur involvieren ausschliesslich fachlich-technische Aspekte. Im Kontext von Software-Wiederverwendung stehen jedoch wie bereits erwähnt auch bestimmte soziale und kulturelle Elemente ([11], S. 2 [12], S.2) in einem Unternehmen. Deshalb sind weiter Massnahmen zu ergreifen, welche auf soziale und kulturelle Aspekte abzielen. An dieser Stelle seien dabei besonders genannt:

- Offenheit
- Kommunikationsfähigkeit
- Kritikfähigkeit
- Vertrauen
- Risikobereitschaft

2.5 Wettbewerbsvorteile

Der Einsatz von Software-Wiederverwendung eröffnet einem Unternehmen Wettbewerbsvorteile, die es ermöglichen, sich langfristig am Markt zu positionieren und sich den sich ändernden Marktbedürfnissen anzupassen. Insbesondere die Wiederverwendung von Komponenten, als eine konkrete Ausprägung von Assets, versprechen hier besondere Vorteile.

Steigerung der Software-Qualität und Zuverlässigkeit
Die Verwendung von Komponenten führt zu einer gesteigerten Qualität und Zuverlässigkeit von Software ([6], S. 2 [11], S. 1). Dies liegt zum einen darin begründet, daß eine Komponente bereits ausführlich getestet wurde und daher für den gewünschten Problembereich leichter eingesetzt werden kann. Zum anderen basieren Komponenten auf bewährten Architektur- resp. Design-Konzepten und tragen so zur Qualitätssteigerung des gesamten Software-Systems bei. Die Zuverlässigkeit steigt selbstverständlich mit der Wiederverwendung der Komponente, da neue Erkenntnisse zu deren Verbesserung resp. Weiterentwicklung führen.

Erhöhte Software-Produktivität
Neben der Software-Qualität und -Zuverlässigkeit läßt sich auch eine Steigerung der Software-Produktivität nennen ([11], S.1 [27] S. 2 [13], S. 1). Indem Applikationen aus Komponenten aufgebaut werden, erhöht sich die Produktivität der Software-Entwicklung, da auf bewährte, getestete Bausteine zurückgegriffen werden kann. Durch Software-Wiederverwendung ist von einer Produktivitätssteigerung um 30% auszugehen. Dies verbessert den Time-to-Market erheblich ([13], S. 2).

Längerfristige Reduktion der Software-Entwicklungs- und Wartungskosten
Als ein weiterer essentieller Vorteil ist längerfristig die Reduktion der Software-Entwicklungs- und –Wartungskosten anzusehen. Jacobson e.a. sprechen von einer Reduktion der Wartungskosten um das 5 bis 10 fache ([14], S.6f.). In diesem Zusammenhang wurde bewußt von längerfristiger Reduktion gesprochen, da die Einführung von Software-Wiederverwendung natürlich mit zunächst relativ hohen Kosten verbunden ist. So geht die Gartner Group davon aus, daß ein mittelgroßes Unternehmen (100 bis 500 Entwickler) zwischen 1,5 Mio. EUR innerhalb von zwei Jahren ausgeben muß, um eine komponentenbasierte Software-Entwicklung zu etablieren (s. Tabelle 2). Der dadurch zu erwartende Return On Investment (ROI) liegt bei min. 200% innerhalb von zwei Jahren ([13], S. 2f.). Andere Quellen sprechen von einer durchschnittlichen Senkung der Entwicklungskosten um 15-75% ([14], S.6f.]). Daran wird deutlich, daß die hohen Ausgaben als eine Investition in die Zukunft betrachtet werden sollten. Die nachfolgende Aufstellung schlüsselt die zu erwartenden Investition in ihre Einzelbestandteile auf.

Benötigtes Element	**Erklärung**	**Niedrige Investion[1] (EUR)**	**Hohe Investion[2] (EUR)**
Asset-Repository-Aufbau	Schaffung eines verwendbaren, zugreifbaren Inventars	59.000	412.000
Asset-Repository Weiterentwicklung und Wartung	20% der Anschaffungskosten	12.000	82.000
Administration, Moderation und Qualitätssicherung	Zu Beginn drei Vollzeitverantwortliche (mit Unterstützung)	265.000	265.000
Belohnungen	20 bis 30 Prozent des Basisgehaltes für teilnehmende Entwickler	294.000	588.000
Gesamt:		630.000	1.347.000

Tabelle 2 : Kosten von Software-Wiederverwendung

Quelle: Eigene Erstellung in Anlehnung an [13], S. 3

Zur Bestimmung der Kosten und des Nutzens von Wiederverwendung können unterschiedliche Metriken und Modelle verwendet werden. In [8], [24] und [26] werden diese ausführlich vorgestellt und diskutiert.

Verbesserte Kundenorientierung
Kundenorientierung ist ein weiterer, wenn auch nicht sofort evidenter, Aspekt der durch den Einsatz von Wiederverwendungstechniken forciert werden kann, da diese die Entwicklung von Prototypen erleichtern ([6], S.2). Durch das sog. Rapid Prototyping wird der Kunde in die Entwicklung des Produktes miteinbezogen und kann somit seine Kritik, seine Verbesserungsvorschläge, aber auch sein Lob anbringen ([28], S.8). Diese Kommunikation über den gesamten Entwicklungsprozess hinweg stellt sicher, daß der

[1] Eigenentwicklung des Asset-Repositories

[2] Kauf eines kommerziellen high-end Asset-Repositories

Kunde mit dem fertigen Produkt zufrieden ist, da er selbst an seiner Entwicklung mitgearbeitet hat und sich dadurch auch damit identifizieren kann.

Bessere Planbarkeit von Projekten
Auch im täglichen Projektgeschäft kann der aus Software-Wiederverwendung resultierende Mehrwert selbstverständlich zu Wettbewerbsvorteilen führen. Neben Produktivitäts-, Qualitäts-, Kosten- sowie Kundenorientierungsaspekten werden auch die Planbarkeit von Projekten und die Zufriedenheit der Projektteammitglieder positiv beeinflußt. Wenn man bspw. bei der Erstellung des Projektplanes aus einem Portfolio von Basislösungen resp. Assets, die für das Projekt wichtigen entnehmen kann, ist man in der Lage die benötigten Ressourcen besser zu planen und so eine realistischere Aussage über den zukünftigen Verlauf des Projektes zu geben. Hierbei ist natürlich die Qualität der Asset-Dokumentation entscheidend. Sie sollte z.B. über die finanziellen Kosten, die Anzahl der zur Implementation benötigten Entwickler und die Interdependenzen zu anderen Assets informieren. Ein auf dieser Grundlage entwickelter Projektplan erhöht auch die Zufriedenheit der Projektteam-Mitglieder, da ihr Einsatz realistischer geplant wurde und sie so nicht überlastet sind.

Erfolgsgeschichten
Des weiteren zeigt sich aus empirischen Beobachtungen, dass sich bei Firmen, die Software-Wiederverwendung einführten, die erwähnten Vorteile einstellten ([6], [14],[27]):
AT&T kann auf ein Wiederverwendungsverhältnis von 40-92% im Bereich Telecoms Operation Support Software verweisen.

- **Ericsson AXE:** 90%tige Wiederverwendung in Hunderten von kundenspezifischen Konfigurationen.
- **Fujitsu** lieferte 70% ihrer Electronic Switching Systems in der vereinbarten Zeit aus. Davor waren es nur 20%.
- Bei **General Accident Insurance** verkürzte sich die durchschnittliche Entwicklungszeit ihrer Systeme um das drei- bis vierfache.
- **Hewlett-Packard** erreichte einen Wiederverwendungslevel von 25-50% in der Firmwareentwicklung für Instrumente- und Drucker.
- **Motorola** verzeichnete eine Wiederverwendungsgrad von 85% und ein 10:1 Produktivitätssteigerungsverhältnis im Compiler- und Compiler-Test Suites Bereich
- **Raytheon** berichtete von einer Produktivitätssteigerung von 50%.
- Die **Navy** beobachtete eine sechsundzwanzigprozentige Reduktion der benötigten Arbeitszeit zur Entwicklung und Wartung ihrer Restructured Tactical Data Systems (RNTDS)

Daran wird evident, dass Software-Wiederverwendung keine pure Theorie ist, sondern vielmehr ein ungemein nützliches Instrument zur Steigerung der Wettbewerbsvorteile eines Unternehmens darstellt.

3 Fazit

Die erfolgreiche Durchführung von Software-Wiederverwendung ist weniger eine Folge des Einsatzes technologischer Mittel als vielmehr eine Folge der Schaffung bestimmter Voraussetzungen im sozialen, kulturellen und organisatorischen Bereich eines Unternehmens.

Erfolgreiche Software-Wiederverwendung kann nur mit einem Kulturwandel innerhalb eines Unternehmen einhergehen. Dieser notwendige Kulturwandel wird dadurch bewirkt, daß sich Menschen in ihren Verhaltensweisen ändern. Diese Änderung geschieht jedoch nicht auf Knopfdruck, sondern es ist ein langwieriger Prozess, der viel Überzeugungsarbeit erfordert und wohlgeplant sein will.

Als in steigendem Maße entscheidente Wettbewerbsfaktoren auf dem IT-Markt können ein möglichst geringes Time-To-Market gepaart mit sich von der Konkurrenz

hervorhebender Qualität genannt werden. Kunden entwickeln zunehmend ein Qualitätsbewusstsein bei der Beurteilung von Software-Systemen. Unternehmen, die heute erkannt haben, daß nur über systematische Software-Wiederverwendung auf die Anforderungen des Marktes positiv reagiert werden kann, werden zu den zukünftigen Gewinnern zählen.

Literaturverzeichnis

[1] Booch, G. (1994): Objektorientierte Analyse und Design, Mit praktischen Anwendungsbeispielen, Addison-Wesley, Bonn 1994

[2] Brown, W. J. e.a. (1998): Anti Patterns, John Wiley & Sons, New York 1998

[3] Bosch, J. e.a. (1996): Object-Oriented Frameworks, Problems and Experiences, Department of Computer Science and Business Administration, University of Karlsrona/Ronneby, Sweden 1996, http:://www.ide.hk-r.se/~ARCS

[4] Buschmann, F. e.a. (1996): Pattern – Oriented Software Architecture. A System of Patterns. Wiley, 1996

[5] Cyperzki, C. (1998): Component Software, Addison-Wesley, Bonn 1998

[6] DOD SRI (1996): Software Reuse Executive Primer, DOD Software Reuse Initiative, Falls Church 1996, http://dii-sw.ncr.disa.mil/ReuseIC/pol-hist/primer/

[7] Fowler, M. (1999): Analysis Patterns. Addison-Wesley, 1999

[8] Frakes W., Terry C. (1996): Software Reuse and Reusability Metrics and Models

[9] Gamma, R. e.a. (1995):Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995

[10] Griss, M. L. (1999): Reuse 2001-2004, What next, now that we have solved all reuse problems?, Hewlett-Packard Laboratories, Palo Alto 1999, http://www.hpl.hp.com/reuse/papers/wisr99-griss.html

[11] Griss, M. L. (1999): Architecting for Large-Scale Systematic Component Reuse, Hewlett-Packard Laboratories, Palo Alto 1999, http://www.hpl.hp.com/reuse/papers/icse99-industrial-griss.html

[12] Griss, M. L. (1999): Domain Engineering and Reuse, Hewlett-Packard Laboratories, Palo Alto 1999, http://www.hpl.hp.com/reuse/papers/ieee99-griss.html

[13] Hunter R. (1997): Gartner View – Once Is Not Enough -, CIO Magazine March 1. 1997http://www.cio.com/archive/030197_garnter_print.html

[14] Jacobson, I. e.a. (1997): Software Reuse. Addison-Wesley, 1997

[15] Kriha W. / Scheffold B. (1998): Soziale Strukturen in neuen Software-Projekten, SYSTOR AG, Zürich 1998

[16] Landin, N. e.a. (1995): Development of Object Oriented Frameworks, Department of Communication Systems, University of Lund, Sweden 1995, http://biblio.die.hk-r.se:8080/~michaelm/fwpages/fwbibl.html

[17] Mattson, M. (1996): Object-Oriented Frameworks, A survey of methodological issues, Thesis, Department of Computer Science and Business Administration, University of Karlsrona/Ronneby, Sweden 1996, http://www.pt.hk-r.se/~michaelm/thesis/toc.html

[18] Monson-Haefel, R. (2000): Enterprise Java Beans. O`Reilly, 2000

[19] Neumann, T. (1998): Präsentation für das OTC CA C++-Forum Komponenten - LEGO für Software Engineers? SYSTOR AG, Zürich 1998

[20] Nowack, P. (1997): Frameworks, Representations & Perspectives, Department of

Computer Science, University of Allborg, Denmark 1997, e-Mail: nowack@cs.auc.dk

[21] Nussbaum, C. (1999): Dauernd am Ball, Wirtschaftswoche Nr. 16, 15.4.1999

[22] Pattison, Ted (2000): Verteilte Anwendungen mit COM+ und Microsoft Visual Basic programmieren., Microsoft Press, 2000

[23] Pree, W. (1997): Komponentenbasierte Softwareentwicklung mit Frameworks. Dpunkt, 1997

[24] Poulin, J. S., Caruso J. M. (1993): Determining the Value of a Corporate Reuse Program, IBM, 1993

[25] SUN (1999): The J2EE Application Programming Model 1999 http://java.sun.com/j2ee/bulletin/apm/apm.html

[26] Wiles, E. (1999): Economical Models of Software Reuse, A survey, comparison and partial validation, v2.1, University of Wales, 1999

[27] Williamson, M. (1997): Software Reuse Introduction, CIO Magazine March 1. 1997,http://www.cio.com/archive/030197_intro_content.html

[28] Williamson, M. (1997): Software Reuse Cultural Issues, CIO Magazine March 1. 1997, http://www.cio.com/archive/030197_cultural_print.html

[29] Williamson, M. (1997): Technology, CIO Magazine March 1. 1997 http://www.cio.com/archive/030197_technology_print.html

[30] Wirfs-Brock, R. e.a. (1990): Designing Object-Oriented Systems, Prentice-Hall Inc., New Jersey 1990