Module und Wiederverwendbarkeit

Ausarbeitung
zum Seminar im Wintersemester 1997/98:

Modulkonzepte in objektorientierten Spezifikationssprachen

Stefan Konst
e-mail: S.Konst@olymp.escape.de

25. November 1997

Inhaltsverzeichnis

Zusammenfassung

Diese Ausarbeitung gibt zunächst eine allgemeine Einführung in die Grundlagen der Wiederverwendung. Danach werden die Prinzipien von Vererbung, Modulen und Entwurfsmustern im Hinblick auf ihre Unterstützung der Wiederverwendbarkeit dargestellt und abschließend miteinander verglichen.

1 Einführung

In der Softwareentwicklung werden objektorientierte Methoden eingesetzt, um die Produktivität und die Wartbarkeit zu verbessern. Dadurch soll die aufgewendete Zeit und somit die Kosten gesenkt werden. Weitere Einsparpotentiale ergeben sich, wenn die erstellten Bausteine bei Bedarf möglichst beliebig oft wiederverwendet werden können. Hierzu müssen während der Entwicklung besondere Anstrengungen unternommen werden, die im folgenden überblickartig dargestellt und verglichen werden sollen. Es ist nicht Thema der Ausarbeitung zu erklären, wie man bestehende Bausteine wiederverwendet, die nicht nach den entsprechenden Kriterien entwickelt wurden.

Als erstes wird auf die allgemeinen Grundlagen der Wiederverwendung eingegangen. Dieser Abschnitt bezieht sich teilweise auf Abschnitt 2 aus [1]. Danach werden die Prinzipien von Vererbung, Modulen und Entwurfsmustern in Hinblick auf ihre Unterstützung der Wiederverwendbarkeit erklärt und schließlich miteinander verglichen. Der Abschnitt über Module bezieht sich dabei auf Abschnitt 3 aus [2], der über Entwurfsmuster auf Abschnitt 3 aus [3] und auf [6].

2 Wiederverwendung

Die Wiederverwendung bestehender Bausteine hat verschiedene Vorteile:

- Allgemeine Erhöhung der Qualität der einzelnen Bausteine, wie auch des gesamten Systems, wenn die Prinzipien für Wiederverwendbarkeit angewandt werden.
- Bereits getestete und verwendete Bausteine enthalten in der Regel weniger Fehler, als neu entwickelte Bausteine.
- Geringerer Entwicklungsaufwand, wenn auf bestehende Bibliotheken zurückgegriffen werden kann. Dadurch erfolgt eine Einsparung von Zeit und Kosten.
- Ein System ist einfacher auf andere Plattformen zu portieren, wenn die einzelnen allgemeinen Bausteine bereits portiert wurden.

Die Qualität einzelner Bausteine, wie auch des gesamten Systems, hängt mindestens von deren Korrektheit, Robustheit, Erweiterbarkeit, Wiederverwendbarkeit und Kompatibilität ab. Die Verbesserung eines Aspekts zieht dabei meistens auch Verbesserungen bei anderen Aspekten nach sich.

Zunächst sollte bei der Entwicklung die Verteilung der Funktionalität auf die einzelnen Bausteine betrachtet werden. Sie sollte so aussehen, daß miteinander

verwandte Funktionen in einem Baustein zusammengefaßt werden und ein Baustein nicht mit Funktionen ausgestattet wird, die nicht seinem eigentlichen Zweck entsprechen.

Weiterhin ist zwischen größerer und geringerer Flexibilität der Bausteine, in Hinblick auf deren eventuelle Wiederverwendung, abzuwägen. Während bei spezialisierteren Bausteinen eher eine geringere Designkomplexität und somit ein geringerer Entwicklungsaufwand notwendig ist, entsteht bei allgemeineren Bausteinen eher eine höhere Designkomplexität und somit ein höherer Entwicklungsaufwand.

2.1 Wiederverwendungsarten

Bei der Softwareentwicklung können folgende Dinge wiederverwendet werden:

- Abgeschlossene Bausteine.
- Die Struktur, in der die Bausteine organisiert sind.

Es hängt von der Abstraktionsebene ab, was wiederverwendet werden kann. So kann die Struktur, welche die Beziehungen zwischen abgeschlossenen Bausteinen beschreibt, auf einer höheren Abstraktionsebene selbst wieder in einem abgeschlossenen Baustein eingekapselt werden.

Nach [1] lassen sich in der objektorientierten Softwareentwicklung grob drei Arten von Wiederverwendung unterscheiden:

- *Black Box Reuse*

  Wiederverwendung von Standardbausteinen, bei denen nur ihre externe Schnittstelle sichtbar und verwendbar ist und für die keine oder nur geringfügige Anpassungen erforderlich sind.

- *Glass Box Reuse*

  Wiederverwendung von zu vervollständigenden oder elementaren Bausteinen, bei denen sowohl ihre interne Struktur, als auch ihre externe Schnittstelle sichtbar sind, aber nur die externe Schnittstelle verwendbar ist.
  Im Rahmen vorgegebener Möglichkeiten werden sie zu neuen Systemen mit den gewünschten Eigenschaften kombiniert oder komplexe Bausteine werden durch „Einschießen“ von Elementarbausteinen neu konfiguriert.

- *White Box Reuse*

  Wiederverwendung von Bausteinen durch Untersuchung und Verwendung sowohl der internen Struktur, als auch der externen Schnittstelle.
  Die Menge vorhandener Bausteine wird um neue komplexe oder Elementarbausteine, z.B. durch Vererbung erweitert, die dann zusammen mit den

vorhandenen Bausteinen zur Konfiguration und Kombination neuer Systeme genutzt werden (Konzept eines Frameworks).

In Tabelle 1 aus [1] werden die soeben vorgestellten Wiederverwendungsarten und deren Zusammenhang mit der Wiederverwendung überblickartig dargestellt.

Art der Wv.	Gegenstand der Wv.	Anpassungs-maßnahme	betrifft primär	Kom-plexität	Dokumenta-tion als	Anpas-sungs-aufwand	Expertise des Nutzers
Black Box Reuse	Standard-bausteine	keine bzw. Methoden-aufrufe oder Parametri-sierung	Instanz	gering	Bibliothek	gering	gering
Glas Box Reuse	komplexe und Elementar-bausteine	konfigurieren kombinieren	Instanz(en)	mittel	quasi als Framework	mittel	mittel
White Box Reuse	Framework	Bausteine erzeugen, konfigurieren und kombinieren	Klasse(n)	mittel bis hoch	Framework	hoch	mittel bis hoch

Tabelle 1: Wiederverwendungsarten

Die hier auf der Ebene einzelner Klassen vorgestellten Zusammenhänge werden, im weiteren Verlauf dieser Ausarbeitung, auf eine höhere Ebene abstrahiert.

3 Vererbung

Auf Klassen angewendet bedeutet konzeptionelle Abstraktion, daß durch Abstraktion aus einzelnen Klassen solche Konzepte herausgelöst, in einer gemeinsamen Superklasse zusammengefaßt und wieder an die Klassen vererbt werden, die in mehreren dieser Klassen vorkommen. Spezielle Klassen könnten allerdings auch von Anfang an durch Vererbung aus einer abstrakten Superklasse abgeleitet werden. Anschaulich kann man ersteres als eine *bottom-up* und letzteres als eine *top-down* Methode beschreiben.

Konzeptionelle Abstraktion ist ein wichtiger Strukturierungsmechanismus für große Softwaresysteme und Bibliotheken. Der Vorteil liegt darin, daß es für ein Konzept nur einen Entwurf und eine Implementierung gibt, was einfach durch Vererbung weitergegeben werden kann. Daher unterstützt Vererbung die Wiederverwendung einzelner Klassen, welche somit, neben anderen Beziehungen, durch eine Vererbungshierarchie miteinander in Beziehung stehen. Ein Nachteil ist jedoch, daß es nicht möglich ist, Invarianten über mehrere Klassen hinweg zu erstrecken, wie dies zum Beispiel mit Hilfe von Modulen erreicht werden kann.

4 Module

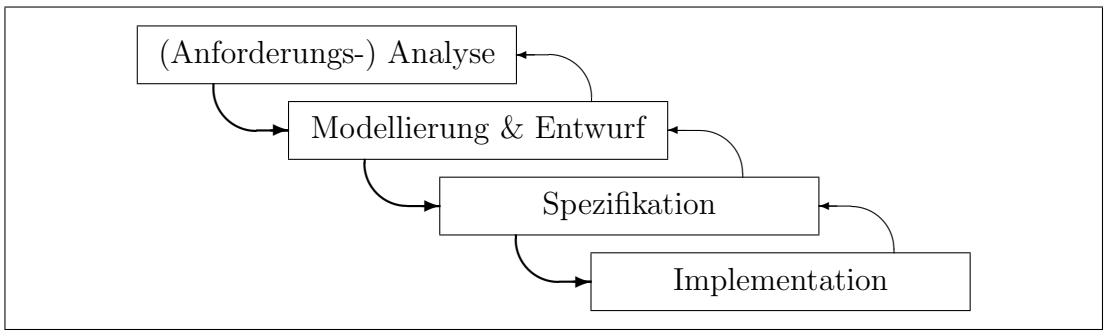
Ein Modul ist eine in sich geschlossene Komponente, die nur über eine eindeutig spezifizierte Schnittstelle mit anderen Komponenten in Verbindung steht, aber nicht in ihrem Inneren mit diesen zusammenhängt.

Daher sollte die Korrektheit eines Moduls ohne Kenntnisse über dessen Einbettung in das Gesamtsystem nachgewiesen werden können. Dies schließt ein, daß in einem aus Modulen aufgebauten System einzelne Module entfernt, hinzugefügt oder ersetzt werden können, ohne daß sich dadurch die Eigenschaften der anderen Module verändern, was eine bessere Wartbarkeit und Erweiterbarkeit zur Folge hat. Weitere Vorteile sind, daß einzelne Module besser verständlich und wiederverwendbar sind und die Verständlichkeit, von aus Modulen aufgebauten Systemen, erhöht wird.

Einzelne Klassen können enger miteinander verbunden sein und dadurch ein Subsystem bilden. Module bieten die Möglichkeit, diese Klassen auf einer höheren Abstraktionsebene zusammenzufassen und das Wissen über die Zusammenhänge der einzelnen Klassen zu verbergen. Dadurch wird „Black Box Reuse“ auf einer höheren Ebene ermöglicht, indem die Abgeschlossenheit wieder hergestellt wird.

4.1 Einkapselung kooperierender Klassen

In [2] wird ein Modulkonzept für objektorientierte Systeme vorgestellt, das im folgenden beschrieben werden soll. Der Autor geht für den Softwareentwicklungsprozeß von folgendem Phasenmodell aus:



Figur 1: Softwareentwicklungsprozeß

Zuerst wird während der (Anforderungs-) Analyse untersucht, was das Gesamtsystem und was einzelne Klassen können müssen. Im nächsten Schritt, der Modellierung und dem Entwurf, werden die Systemarchitektur, die Zusammenhänge und Leistungsvereinbarungen zwischen den Klassen und die Subsysteme festgelegt. In der Spezifikationsphase werden die Leistungsvereinbarungen zwischen den Klassen durch öffentliche Methoden und Subsysteme kooperierender

Klassen durch Module spezifiziert. Abschließend wird während der Implementierung abstrakter Pseudocode in ausführbaren Code umgesetzt.

Nach [2] müssen Module bei der Softwareentwicklung folgende grundlegende Kriterien erfüllen, damit sie später problemlos wiederverwendet werden können. Diese Kriterien dienen auch dazu zu entscheiden, welche Klassen so eng voneinander abhängen, daß sie in einem Modul eingekapselt werden müssen.

Jedes Modul ist mit einer präzise definierten Schnittstelle zu versehen und modulweite Invarianten sind zu berücksichtigen. Invarianten dienen dazu, die Anzahl der möglichen Zustände einer Klasse einzuschränken oder die Beziehungen zwischen mehreren Objekten zu beschreiben.

Die Kriterieren 1, 2 und 3 sind bei der Konstruktion zu beachten. Die Kriterieren 4 und 5 sind Bedingungen, die die Konsistenz von Modulen garantieren, das heißt, daß alle Invarianten immer halten.

Kriterium 1: Ein Modul besteht aus mehreren Klassen. Es hat eine wohldefinierte Schnittstelle. Diese Schnittstelle ist eine Untermenge der Schnittstellen (öffentlichen Methoden) der in ihm enthaltenden Klassen. Sie enthält solche Methoden, die nicht nur für andere Klassen innerhalb des Moduls, sondern auch für Klassen außerhalb des Moduls verfügbar gemacht werden.

Kriterium 2: Es gibt modulweite Invarianten, die einzelnen Klassen zugeordnet sind. Sie können auf beliebige Instanzen von Klassen innerhalb des Moduls verweisen.

Kriterium 3: Die modulweiten Invarianten verweisen nicht auf Klassen außerhalb des Moduls.

Kriterium 4: Alle exportierten, nichterzeugenden Methoden bewahren alle Invarianten des Moduls.

Kriterium 5: Alle exportierten, erzeugenden Methoden richten alle Invarianten des Moduls ein.

Das erste Kriterium folgt unmittelbar aus den Prinzipen für die Einkapselung. Dabei sollte die Schnittstelle nach [4] folgende Anforderungen erfüllen:

- *Wenige Schnittstellen*

  Module sollten nur mit wenigen anderen Modulen kommunizieren (müssen).

- *Kleine Schnittstellen*

  Wenn zwei Module miteinander kommunizieren, sollten sie nur so wenig wie möglich Informationen miteinander austauschen.

- *Explizite Schnittstellen*

  Der Informationsfluß sollte an der Schnittstelle gut sichtbar sein und nicht etwa über globale Variablen geschehen, was schlecht nachvollziehbar ist.

Das zweite Kriterium folgt aus der Tatsache, daß die Spezifikation von Beziehungen zwischen kooperierenden Klassen Invarianten benötigt, die für all diese Klassen gelten.

Die Einschränkungen für Module bei den Kriterien 3, 4 und 5 stammen von der Notwendigkeit, die Konsistenz von Modulen zu überprüfen. Das heißt, es ist zu garantieren, daß alle Invarianten immer halten.

An die Konsistenz werden zwei Anforderungen gestellt:

1. Konsistenzüberprüfungen müssen lokal sein.
2. Wenn sie für eine Komponente erreicht wurde, darf sie nicht durch eine andere Komponente verletzt werden.

Bei der Konsistenzüberprüfung müssen die Invarianten betrachtet werden. Wenn eine Methode ein Objekt verändert, muß sie die Invariante bewahren, oder sollte ein Objekt durch eine Methode erzeugt werden, muß die Methode auch die Invariante einrichten.

Nach der ersten Anforderung muß die Untersuchung der Methoden nicht über das gesamte System hinweg erfolgen. Dies wird durch Kriterium 3 garantiert.

Um die zweite Anforderung zu erfüllen muß sichergestellt werden, daß Invarianten nicht von außerhalb des Moduls, verletzt werden können. Dies könnte bei Aufrufen von öffentlichen Methoden einer Klasse des Moduls geschehen. Durch Erfüllung der Kriterien 4 und 5 können Verletzungen ausgeschlossen werden. Dazu muß der Export von Methoden beschränkt oder verboten werden.

4.2 Kompatibilität

Durch Kompatibilität zwischen Klassen kann ein Objekt einer Superklasse durch ein Objekt einer Subklasse ersetzt werden. Kompatibilität kann durch Vererbung ausgedrückt werden, weil eine Subklasse mindestens die Eigenschaften ihrer vererbenden Superklasse hat.

Nach [2] kann Kompatibilität zwischen Klassen auf Kompatibilität zwischen Modulen erweitert werden. Dadurch ist es möglich, die konzeptionelle Abstraktion bei Klassen auf Module zu erweitern und somit zur Skalierbarkeit konzeptioneller Abstraktion beizutragen.

Für die Kompatibilität zwischen Modulen sind folgende Kriterien zu erfüllen:

Kriterium 6: Die Schnittstelle eines kompatiblen Submoduls ist nicht schmaler, als die Schnittstelle des Supermoduls.

Zwischen den Schnittstellen muß gelten, daß es zu jeder exportierten Methode des Supermoduls eine korrespondierende exportierte Methode des Submoduls gibt, mit Vorbedingungen, die nicht stärker, und Nachbedingungen, die nicht schwächer sind.

Kriterium 7: Die Invarianten eines kompatiblen Submoduls sind nicht schwächer, als die Invarianten des Supermoduls.

An die einzelnen davon betroffenen Klassen gibt es keine speziellen Anforderungen mit Ausnahme der Methoden, die von ihnen exportiert werden. Weiterhin kann es noch mehr Beziehungen zwischen Klassen des Supermoduls und Klassen des Submoduls geben und ein kompatibles Submodul kann zusätzliche Klassen beinhalten.

Kompatibiltät zwischen Modulen bringt verschiedene Vorteile:

- Durch die Erweiterung von konzeptioneller Abstraktion von Klassen auf Module, kann diese auf einer höheren Ebene stattfinden, wodurch Strukturen von großen Systemen verständlicher werden.
- Kompatibilität zwischen Modulen begünstigt deren Austauschbarkeit. Alle Vorkommen einer Klasse des Supermoduls können durch eine kompatible Klasse des Submoduls ersetzt werden, wenn die *Create* Methode der Klasse des Supermoduls mit der *Create* Methode der Klasse des Submoduls korrespondiert.
- Einzelne Klassen können im Zusammenhang von verschiedenen Modulen benutzt werden. Dadurch werden Module davor geschützt, Softwaresysteme oder Bibliotheken in streng abgetrennte Bereiche einzuteilen. Denn eine Einteilung eines Systems in sich nicht überlappende Module ist in der Praxis schwer zu erreichen.

  Dadurch, daß die Kriterien für Kompatibilität eingehalten werden, ist es problemlos möglich, verschiedene Versionen einer Klasse in mehreren Modulen zu haben.

5 Entwurfsmuster

Entwurfsmuster (Design Patterns) beschreiben allgemeine Strukturen und geben eine Anleitung, wie einzelne Komponenten zu einem Gesamtsystem zusammengefügt werden können. Sie ermöglichen es somit, nicht nur die Implementation, sondern auch das Design wiederzuverwenden.

Sie entwickeln sich weiter, indem bei ihrer Anwendung Feinheiten gefunden werden, die bei ihrer nächsten Anwendung berücksichtigt werden können. Daher repräsentieren sie für ein Problem eine Lösung, die bereits erfolgreich angewendet wurde.

Nach [6] bestehen Entwurfsmuster aus vier grundlegenden Elementen:

- *Entwurfsmustername*

  Er sollte bereits einen ersten Eindruck über die Möglichkeiten des Entwurfsmusters vermitteln.

- *Problemabschnitt*

  Hier wird beschrieben, wann und für welche Probleme das Entwurfsmuster anzuwenden ist, in welchem Zusammenhang es zu sehen ist.

- *Lösungsabschnitt*

  Hier werden die verwendeten Elemente beschrieben, sowie deren Beziehungen, Zuständigkeiten und Interaktionen untereinander. Er liefert eine Schablone, die in verschiedenen Situationen angewendet werden kann. Dies ist weder ein bestimmter Entwurf noch eine konkrete Implementierung, sondern eine abstrakte Beschreibung, wie eine allgemeine Anordnung von Elementen ein Entwurfsproblem löst.

- *Konsequenzenabschnitt*

  Hier werden die Vor- und Nachteile des Entwurfs aufgelistet. Sie dienen dazu, diesen Entwurf mit Alternativen zu vergleichen, die zum Beispiel ein anderes Zeit- und Speicherplatzverhalten haben. Er enthält weiterhin Informationen über seinen Einfluß auf die Flexibilität, Erweiterbarkeit und Portierbarkeit des Gesamtsystems.

Entwurfsmuster werden in Katalogen organisiert, aus denen sie bei Bedarf ausgewählt werden können. Die Beschreibung eines Entwurfsmusters in solch einem Katalog erfolgt verbal, wobei die verschiedensten Aspekte berücksichtigt werden. Die grafische Darstellung eines Entwurfsmusters stellt letztendlich als Endergebnis des Entwurfprozesses dar, welche Beziehungen zwischen Objekten und Klassen herrschen.

In [6] werden zum Beispiel folgende Aspekte von Entwurfsmustern beschrieben:

Mustername und Klassifizierung	*Zweck*	*Auch bekannt als*
Motivation	*Anwendbarkeit*	*Struktur*
Teilnehmer	*Interaktionen*	*Konsequenzen*
Implementierung	*Beispielcode*	*Bekannte Verwendungen*
Verwandte Muster		

Nach [3] müssen Entwurfsmuster nach folgenden Prinzipien konstruiert werden, um die problemlose Wiederverwendung zu ermöglichen:

- *Abstraktion*

  Vererbung muß eine Kompatibilitätsbeziehung zwischen Objekten oder Klassen darstellen. Obwohl es vom speziellen Entwurfsmuster abhängt, welche Art von Kompatibilität erforderlich ist, muß meistens Konformität zwischen den Klassen gelten, die voneinander erben.

- *Einkapselung*

  Systementwurf mit Hilfe von Entwurfsmustern bedeutet, Systeme aus mehreren Mustern zusammenzustellen. Damit diese Zusammenstellung und das resultierende System modular wird, ist es wichtig, daß einzelne Entwurfsmuster in sich geschlossen sind. Einkapselung verringert Abhängigkeiten zwischen Entwurfsmustern, wenn sie miteinander verbunden werden und steigert dadurch die Flexibilität bei der Zusammenstellung.

- *Spezifikation*

  Die Schnittstelle eines Entwurfsmusters muß spezifiziert werden. Entwurfsmuster sollten als wohleingekapselte Komponenten mit klar definierter Schnittstelle gesehen werden.

Entwurfsmuster verwenden Vererbung, um Kompatibilität auszudrücken. Diese macht Komponenten zuverlässiger.

Zur Spezifikation eines wohleingekapselten Musters muß entschieden werden, welche Methoden lokal und welche global sein sollen. Nach außen verfügbar sollten nur solche Methoden sein, die wirklich nötig sind.

Die Kommunikation mit der Außenwelt sollte nur über ein spezielles Kommunikationsobjekt erfolgen, das heißt, daß Leistungen nur über dieses Objekt abrufbar sein sollten. Das Entwurfsmuster *Glue* beschreibt dieses Prinzip im allgemeinen, es kommt aber auch in anderen vor.

Ein Framework ist eine Menge abstrakter und daraus abgeleiteter spezieller Klassen. Nach [5] unterstützen Entwurfsmuster nicht nur die Entwicklung von Systemen, die auf Frameworks beruhen, sondern ermöglichen auch die Entwicklung neuer Frameworks anhand der Strukturen bei der Entwicklung anderer Frameworks.

5.1 Anwendung

Entwurfsmuster beschreiben komplexe Zusammenhänge. Daher kann es schwierig werden, für ein bestimmtes Problem sofort das passende Muster zu finden, da zuviele Parameter berücksichtigt werden müssen.

Nach [6] sollte bei der Auswahl eines Entwurfsmusters folgendes bedacht werden:

- Betrachtung, wie Entwurfsmuster Entwurfsprobleme lösen.
- Vergleich der Zweckabschnitte.
- Beziehungen zwischen den Entwurfsmustern.
- Untersuchung von Entwurfsmustern mit gleicher Aufgabe.
- Untersuchung der Gründe für Entwurfsrevisionen.
- Betrachtung, was am Entwurf geändert werden könnte.

Die Anwendung eines Entwurfsmusters sollte dann folgendermaßen geschehen:

1. Das gesamte Muster durchlesen, um einen Überblick zu gewinnen.
2. Verstehen der Klassen und Objekte des Musters und deren Beziehungen untereinander.
3. Betrachtung des Beispielcodes, um ein konkretes Beispiel für die Implementierung des Musters zu sehen.
4. Auswahl der Namen für die Klassen und Objekte des Musters, die im eigenen Anwendungskontext sinnvoll sind.
5. Definition der Klassen.
6. Definition von anwendungsspezifischen Namen für Operationen im Muster.
7. Implementierung der Operationen so, daß die im Muster festgelegten Zuständigkeiten und Interaktionen umgesetzt werden.

Natürlich kann man diese Vorgehensweise im Laufe der Zeit den eigenen Erfahrungen anpassen. Entwurfsmuster sollten nicht wahllos eingesetzt werden, sondern nur da, wo dies im Hinblick auf die Flexibilität auch wirklich nötig ist.

In [6] wird eine gute Einführung in Entwurfsmuster und ihre Anwendung gegeben. Der Hauptteil umfaßt einen Katalog mit verschiedenen grundlegenden Mustern.

6 Wiederverwendbarkeit — ein Vergleich

Die objektorientierte Softwareentwicklung ermöglicht drei Stufen von Wiederverwendung.

- Wiederverwendung einzelner Klassen durch Vererbung.
- Wiederverwendung der Struktur, in der die Klassen organisiert sind, durch Entwurfsmuster.
- Wiederverwendung eingekapselter Klassen und deren Beziehungen zueinander durch Module.

Klassen und Module bilden abgeschlossene Bausteine. Daher können sie durch „Black Box Reuse“ wiederverwendet werden. Während mit Hilfe eines Moduls ein bestimmter Entwurf und dessen Implemeniterung wiederverwendet werden kann, kann bei Entwurfsmustern ein allgemeiner Entwurf wiederverwendet werden. Daher sind Entwurfsmuster auf der Ebene von „Glas Box Reuse“ und „White Box Reuse“ anzusiedeln.

Im Sinne der Einkapselung kooperierender Klassen sind Entwurfsmuster nach [3] spezielle Module. Softwaresysteme können durch Module und Entwurfsmuster komponentenweise aufgebaut werden. Der Unterschied zwischen beiden liegt darin, daß Module sofort verwendet werden können und Entwurfsmuster nur eine Anleitung zur Implementierung der Komponente geben. Bei der Implementierung eines Entwurfsmusters sollte ein Modul entstehen.

Um bessere Softwaresysteme zu erhalten, sollten sie modular aus wiederverwendbaren Komponenten aufgebaut werden.

Literatur

[1] Ulrich W. Eisenecker. *Objektorientierte Software wiederverwendbar entwerfen.*

[2] Andreas Rüping. *Modules In Object-Oriented Systems.*

[3] Thomas Lindner, Andreas Rüping. *How Formal Object-Oriented Design Supports Reuse.*

[4] Andreas Heuer. *Objektorientierte Datenbanken.* Addison-Wesley, 1997.

[5] Wolfgang Pree. *Design Patterns for Object-Oriented Software Development.* Addison-Wesley, 1994.

[6] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Entwurfsmuster.* Addison-Wesley (Deutschland), 1996.