

Objektorientierte Entwurfsmethodik

Christian Viehmann
Student der Universität Stuttgart
christian.viehmann@gmx.de

Überblick

Das Thema der objektorientierten Entwurfsmethodik ist sehr weitreichend und umfangreich. Diese Abhandlung beinhaltet zuerst einmal eine Einführung und eine Klärung der Begriffe. Dann beschäftigt sie sich mit einigen wichtigen Methoden, zwei spezielle werden eingehender behandelt. Danach werden noch interessante Ansätze vorgestellt, die nicht als eigene Methoden gesehen werden können, doch bei Verwendung anderer Methoden sehr nützlich sind. Der Schwerpunkt am Schluss liegt auf einer Zusammenfassung einiger besonders interessanter Methoden.

1. Einführung

Gerade beim objektorientierten Entwurf ist es wichtig, am Schluss ein klar strukturiertes und schlüssiges Konzept des Programms zu erhalten. Die Klassen und Vererbungen sollten logisch zusammenhängen und wohldefiniert sein. Die Kopplung zwischen Komponenten sollte möglichst gering sein, der Zusammenhalt innerhalb einer Komponente möglichst hoch. Das sind wohlbekannte Richtlinien für einen guten Entwurf. Aber wie fängt man an? Was kommt zuerst, was macht man am Schluss? Gibt es Verfahren, die helfen, diese Richtlinien einzuhalten?

Mit dieser Problematik beschäftigt sich die objektorientierte Entwurfsmethodik. Die Thematik wird schnell noch klarer, nachdem die wichtigen Begriffe umrissen sind.

2. Begriffsklärung

Es gibt mehrere relevante Begriffe, die im Zusammenhang mit Entwurfsmethodik stehen und hier noch einmal kurz geklärt und gegeneinander abgegrenzt werden sollen. Zuerst einmal steht der Begriff „**Methodik**“ selbst im Raum. Methodik, auch **Methodologie** oder Methodenlehre genannt, ist die Lehre von den Wegen wissenschaftlicher Erkenntnis und von den Verfahrensweisen der Wissenschaft. Dabei enthalten ist auch zugleich eine Anleitung zu planmäßigem wissenschaftlichen Vorgehen [1]. Vereinfacht gesagt ist die Methodik die Lehre von den Methoden.

Dieser zweite, sogar wortverwandte Begriff wird folgendermaßen definiert: Eine **Methode** ist ein planmäßiges

Verfahren, das zu technischer Fertigkeit bei der Lösung theoretischer und praktischer Aufgaben führt [1]. Bei Entwurfsmethoden geht es um ein Verfahren, zur Strukturierung eines objektorientierten Programms.

Ein weiterer Begriff sollte von den oben genannten abgegrenzt werden, und zwar einer der vorhergehenden Themenschwerpunkte, die „**Entwurfsprinzipien**“. Prinzipien legen etwas Grundsätzliches fest. Sie bestimmen, wie etwas beschaffen sein oder aussehen soll. Im Gegensatz dazu stehen bei der Methode die Fragen „Welche Tätigkeiten müssen nacheinander ausgeführt werden?“ oder „Wie packe ich es an?“ im Vordergrund.

Da einige Leser sich bestimmt wundern, wieso das Wort UML (Unified Modeling Language) nicht fällt, wird hier kurz erklärt, warum UML nicht zu den Methoden gezählt wird. UML ist, wie der Name schon sagt, eine Sprache. Mit einer Sprache werden Sachverhalte beschrieben oder definiert. Mit UML kann zum Beispiel eine Klassenstruktur oder kompletter Entwurf beschrieben werden. Eine Sprache gibt aber keine Anleitung oder stellt kein Verfahren dar, wie eine Aufgabe gelöst wird. Da das aber von einer Methode zwingend gefordert wird, ist UML keine Methode.

3. Einige gängige Entwurfsmethoden

Ein paar der 21 Entwurfsverfahren, die Hutt beschreibt [2], sind hier kurz angerissen. Dabei wurden vor allem die in der Literatur am häufigsten referenzierten gewählt. Den beiden Ansätzen 3.1 Booch-Methode und 3.4 OOA, OOD & OOP sind, im Verlauf des Dokumentes jeweils eigene Kapitel gewidmet, das heißt sie werden noch eingehender behandelt.

3.1. Booch-Methode

Diese Methode stammt, wie der Name schon erkennen lässt, von Grady Booch und entstand um 1994. Das Verfahren beinhaltet auch einen Vorläufer von UML zu Analyse- und Entwurfszwecken. Booch teilt den Entwurf in die zwei Phasen „Domain Analysis“ und „Design“. In der ersteren wird eine Klassenstruktur erstellt, in der zweiten Phase kommt die spezielle Umgebung zum Tragen, wie zum Beispiel Betriebssystem oder Datenbanktyp. Auf diese Methode wird unter 4. Booch-Methode noch näher eingegangen.

3.2. Objectory

Objectory wurde von Ivar Jacobson 1987 aufgrund seiner Erfahrungen bei Ericsson ausgearbeitet. Die Methode wurde immer weiterentwickelt und erschien 1992 als Buch. 1998 diente die Version 3.8 des Verfahrens als Basis für den Unified Rational Process von Rational Software.

Es basiert auf Use-Cases und stellt die Frage, wie ein System tatsächlich benutzt wird, in den Mittelpunkt des Prozesses. Indem die Analyse- und Entwurfsmodelle unter Berücksichtigung von Benutzungsszenarien entwickelt werden, sind die damit entwickelten Programme robuster und besser bedienbar. Jacobson schlägt ein Vorgehen in fünf Phasen, von der Analyse bis zum Test, vor. Dabei können Phasen iteriert werden und sich überlappen. In der Entwurfsphase wird versucht, die Analyseergebnisse, hier Use-Case Diagramme, durch Regeln und Vorgänge auf eine Klassen- und Objektstruktur abzubilden. Dabei entsteht zum Beispiel für jeden Use-Case ein Interaktionsdiagramm (oder auch Kollaborationsdiagramm genannt).

3.3. OMT (Object Modeling Technique)

Dieses Verfahren wurde von Rumbaugh, Blaha, Premerlani, Eddy und Lorensen um 1991 entwickelt. Es beinhaltet Schritt-für-Schritt-Anleitungen für Prozesse, ein Index der Konzepte und Diagrammregeln. Der Softwareprozess wird in vier Phasen aufgeteilt: Analyse, Systementwurf, Objektentwurf und Implementierung. Dabei konzentriert man sich beim Systementwurf auf die gesamte Architektur, während man beim Objektentwurf nur das Objektmodell optimiert und verfeinert. Die Systementwurfsphase besteht hier aus 8 Schritten, wobei verschiedene Schritte auch Unterschritte beinhalten können.

3.4. OOA, OOD & OOP

Die Namen Coad, Yourdon und Nicola stehen für diese Methode, die 1991 erstmals publiziert wurde. Es gibt eine Vielzahl an Kursen und Material dazu, vor allem von Object International, Inc. OOA heißt Object-Oriented Analysis, OOD Object-Oriented Design und OOP, nicht mehr schwer zu erkennen, Object-Oriented Programming. Die Begriffe OOA, OOD und OOP werden auch allgemein benutzt. In diesem Dokument beziehen sie sich immer auf die Methode von Coad, Yourdon und Nicola. Die Verfasser teilen jeder Phase Aktivitäten zu, die nicht unbedingt in einer festgelegten Reihenfolge abgearbeitet werden müssen. Das unterscheidet das Konzept deutlich von den vorhergehenden. Ferner liegt zu jeder Aktivität ein Katalog von Fragen vor, die den Entwickler beim Modellieren führen oder auf häufige Schwierigkeiten hinwei-

sen. Auch diese Methode wird in Kapitel 5 OOA, OOD & OOP näher beleuchtet.

3.5. SSADM

SSADM bedeutet "Structured Systems Analysis and Design Methodology". Die in der Datenbankmodellierung wohlbekannte Teilung von externen Schemata, internem Schema und konzeptionellem Schema nach ANSI-SPARC wird bei SSADM auf objektorientierte Programme übertragen. Dafür stehen, ähnlich wie in UML, verschiedene Diagramme, die Objekte und deren Beziehungen untereinander darstellen können, zur Verfügung. Dabei bleibt aber die Dreiteilung zentraler Aspekt. Darüber hinaus gibt es Techniken und Anleitungen mit über 1000 Schritten, um dem Entwickler diese Vorgehensweise untergliedert anzubieten.

4. Booch-Methode

Die Booch-Methode stellt sowohl einen Prozess, als auch eine Notation, um den Entwurf eines Softwaresystems zu entwickeln, bereit. Booch identifizierte zuerst vier fundamentale Sichten, wie eine Software betrachtet werden kann: die dynamische Sicht, die logische Sicht, die statische Sicht und die physische Sicht. Die Notation trägt dem Rechnung, indem sie jede der Sichten durch bestimmte Diagramme unterstützt. Dieses Konzept der Diagramme und Sichten ist in den UML-Standard übernommen und weiterentwickelt worden.

Die Sichten und die Notation waren zweifellos durchdachte und bahnbrechende Entwicklungen, doch wie Eingangs schon erwähnt, sind Notationen und Sprachen keine Methoden. Deshalb bleibt die Notation hier auch nur so kurz geschildert und der oft vernachlässigte Teil der Booch-Methode, der Prozess, wird in den Vordergrund gestellt.

In [3] ist noch eine Vorgängerversion des jetzigen Prozessmodells beschrieben. Damals gab es einen Makroprozess, bestehend aus Konzeption, Analyse, Entwurf, Evolution und Wartung, in dem jeweils gleiche Mikroprozesse abliefen. Diese Vorgehensweise ist dem Spiralmodell von Boehm [4] nachempfunden. In der heutigen Version führt Booch nur noch drei Phasen an, die jedoch auch iterativ inkrementell gedacht sind. Man kann zum Beispiel vom Entwurf noch einmal zur Analyse zurückkehren. Die Idee ist, zuerst die Kern- oder Hauptfunktionen eines Systems zu identifizieren und gleich anschließend diese Funktionen zu entwerfen und in einem Prototyp zu implementieren. Danach wird das System in weiteren Iterationen verfeinert. Folgende Phasen sind beschrieben:

- Anforderungsanalyse (Requirements Analysis),
- Domain-Analyse (Domain Analysis) und

- Entwurf (Design).

Die drei Phasen werden im Folgenden in Unterkapiteln erläutert. Das Hauptaugenmerk liegt dabei auf den letzten zwei Phasen, da sie den Entwurf behandeln.

4.1. Anforderungsanalyse

Den Analyseprozess lässt Booch recht frei. Er gibt keine formalen Schritte an und begründet dies mit der hohen Varianz des Prozesses. Der Prozess variiert, je nachdem, ob das System neu ist, ob und wie motiviert der Kunde zur Mitarbeit bereit ist und ob es schon Dokumente über den Sachverhalt gibt, den das Programm automatisieren soll.

Als Produkte der Analysephase entstehen bei Booch im Wesentlichen zwei Ergebnisse. Als erstes werden die Kern- oder Hauptfunktionen der zu erstellenden Software ermittelt. Das zweite ist eine Menge von Hauptmechanismen (Key Mechanisms). Die Spezifikation eines Hauptmechanismus beinhaltet einen Eingabestatus, einen Ausgabestatus und erwartete Statusänderungen. Hier hat man also ein ähnliches Vorgehen wie bei einem Automaten.

4.2. Domain-Analyse

Dieses Gebiet der Analyse hat es zur Aufgabe ein kurzes, prägnantes und objektorientiertes Modell eines Teils der echten Welt zu erstellen. Typischerweise besteht dieser Teil aus den, für die Software relevanten, Bereichen des Unternehmens des Kunden.

Die folgenden Schritte sind hochgradig iterativ zu betrachten, denn manchmal erkennt man Attribute erst, wenn man gerade Vererbungen festlegt. Man kann also die Schritte mehrmals nacheinander ausführen oder zu einem Schritt zurückkehren, den man schon bearbeitet hat. Folgende Schritte werden bei der Domain-Analyse ausgeführt:

4.2.1. Klassen definieren. Zuerst werden nur die Hauptklassen identifiziert, in einer späteren Iteration kann dann verfeinert werden. Hier schlägt Booch die Erstellung eines sogenannten „Data Dictionary“, also eine Art Begriffslexikon vor. In diesem Lexikon wird alles festgehalten und weiterentwickelt, was für das Objektmodell interessant sein könnte. Es wird zuerst einmal alles aufgenommen, was relevant sein könnte. Stellt sich später heraus, dass man einen Eintrag doch nicht benötigt, wird er wieder gestrichen. Bei diesem Vorgang wird auch auf die CRC-Karten, die in Kapitel 6.2. vorgestellt werden, als Technik zur Erkennung und Strukturierung von Klassen verwiesen.

4.2.2. Beziehungen finden. Es geht dabei um die Beziehungen zwischen den Klassen, zuerst einmal um die Beziehungen der Hauptklassen. Vor allem Assoziationsbeziehungen werden in diesem Schritt entdeckt. Es kann nämlich auch sein, dass sich eine Assoziation später als eine Vererbung herausstellt. Das passiert jedoch erst im 4. Schritt.

Um die Beziehungen zu identifizieren schlägt Booch vor, immer eine kleine Menge an Klassen aus einer bestimmten Sicht oder aus einem Szenario zu betrachten. Beim Durchspielen des Szenarios oder beim paarweisen Abgleich wird ermittelt, ob die eine Klasse von der anderen abhängig ist. Ist dies der Fall, wird eine Assoziationsbeziehung zwischen den beiden betreffenden Klassen eingeführt.

4.2.3. Attribute finden. Es gibt zwei Ansätze. Erstens kann zu jeder Klasse bestimmt werden, welche Daten sie benötigt. Zweitens kann man von den benötigten Daten auszugehen und zu schauen, zu welcher Klasse sie am besten passen. Auch hier werden CRC-Karten als Hilfsmittel aufgeführt.

4.2.4. Vererbung definieren. Das bedeutet, es werden Generalisierungs- und Spezialisierungsbeziehungen aufgedeckt. Dabei stützt man sich auf die bereits vorhandenen Assoziationen. Die Klassen werden in einen schon bestehenden Vererbungsgraph eingebunden oder es wird ein neuer erstellt. Eine weitere Technik um Vererbungen zu definieren ist, Objekte mit ähnlichem Verhalten, die noch zusammenhangslos sind, von einem gemeinsamen Objekt alle Ähnlichkeiten erben zu lassen.

4.2.5. Operationen festlegen. Operationen sind Tätigkeiten, die eine Klasse ausführen kann. Sie werden auch Methoden genannt. Zuerst versucht man nur die Hauptoperationen zu bestimmen, die das System benötigt, um die Anforderungen zu erfüllen. Später können besonders umfangreiche Operationen in mehrere untergliedert werden, um die Übersichtlichkeit zu wahren.

4.2.6. Validierung und Iteration. Das bedeutet vor allem, das entstandene Modell kritisch zu überdenken und an Szenarien auszutesten. Das Modell wird dann repariert oder erweitert, sodass die gefundenen Unstimmigkeiten beseitigt sind. Diese Schritte der Validierung können mehrmals durchlaufen werden, bis die gewünschte Feinheit erreicht ist.

4.3. Phase: Entwurf

Der Entwurf stellt die physikalische Struktur des Programms bereit, bildet die logische Struktur darauf ab und führt zu lauffähigen Prototypen. Das heißt, der Entwurf

bildet die logische Konzeption aus der Domain-Analyse auf die physikalische, also auf die wirkliche Umgebung ab, wie zum Beispiel Programmiersprache, verwendete Prozessoren oder Ausgabegeräte.

4.3.1. Architektur bestimmen. Das bedeutet bei Booch die Festlegung auf eine Programmiersprache, darauf welche Prozessoren benutzt werden, auf welchem Betriebssystem die Software laufen soll und so weiter. Außerdem gehört zu diesem Schritt die Planung, für welche Hauptaufgaben der Software Prototypen entwickelt werden sollen.

4.3.2. Logischen Entwurf bestimmen. Hier werden alle Datentypen spezifiziert und die Operationen (beziehungsweise Methoden) feiner ausgearbeitet. Zusätzlich wird entschieden, welche Teile des Objekts „private“ bleiben und welche „public“ werden sollen.

4.3.3. Abbildung auf physische Implementierung. In diesem Schritt spielt die physische Sicht die Hauptrolle. Das heißt die Schnittstellen zum Betriebssystem und dabei auch zu Ein- und Ausgabegeräten werden identifiziert und es wird entschieden, welche logischen Aus- und Eingaben über welche Schnittstellen erfolgen sollen. Manchmal ist es sogar erforderlich, eine Zwischenschicht einzuführen, deren Klassen helfen, die Kommunikation zwischen Peripheriegeräten und Software zu erleichtern.

4.3.4. Verfeinerung. Hier werden die mit den Prototypen gesammelten Erfahrungen ausgewertet und der Entwurf dementsprechend modifiziert, beziehungsweise erweitert.

4.4. Bewertung

Booch war einer der ersten, die sich mit der Problematik des objektorientierten Entwurfs beschäftigten. Viele Abläufe seiner Methode erscheinen uns heute selbstverständlich, doch damals waren sie fast revolutionär. Vor allem das iterative und inkrementelle Vorgehen seiner Methode und die Notation haben zu dem großen Erfolg seiner Methode geführt. Viele Teile der Methode haben Einfluss auf die Sprache UML und den Rational Unified Process gehabt. Die Booch-Methode ist eine der grundlegenden Methoden der Objektorientierung.

5. OOA, OOD & OOP

Da die Methode schon in ihrem Namen geteilt ist, ist es klar, dass es drei Bereiche gibt. Diese Bereiche haben folgende Zuständigkeiten:

- OOA (Object-Oriented Analysis) arbeitet mit Klassen und Objekten der echten Welt. Das heißt

hier wird versucht, die Wirklichkeit auf Klassen und Objekte abzubilden.

- OOD (Object-Oriented Design) konzentriert sich auf Mensch-Rechner-Interaktionen, Taskmanagement und Klassen und Objekte zur Verwaltung der Daten.
- OOP (Object-Oriented Programming) bietet Strategien zur systematischen Implementierung der Ergebnisse von OOA und OOD.

Wie auch schon bei Booch beinhaltet die Methode eine eigene Notation zur Darstellung der Objekt- und Klassenstruktur. Dabei wird zwischen fünf Detaillierungsgraden unterschieden, die Schichten (Layers) genannt werden. Die erste Schicht beinhaltet nur Gruppen von Hauptklassen, ähnlich Paketen. Die letzte Schicht bietet den höchsten Detaillierungsgrad und zeigt alle relevanten Informationen wie Attribute, Methoden und Beziehungen zwischen Klassen.

Die Architektur der Software wird von vornherein in vier Komponenten geteilt: eine für Benutzerinteraktionen, eine für Datenverwaltung, eine für die Echte-Welt-Klassen und eine für Taskmanagement. Dieses Vorgehen erinnert stark an das Model-View-Controller-Muster (MVC-Pattern).

5.1. Prozessmodell

Die Methode bietet ein eigenes Prozessmodell, das Baseball-Modell. Ferner werden noch verschiedene Aspekte behandelt, die sich auf den Prozess auswirken, wie das Arbeiten in interdisziplinären Teams.

5.1.1. Baseball-Modell. OOA, OOD und OOP sind laut Coad, Yourdon und Nicola keine Schritte oder Phasen, sondern Aktivitäten. Das heißt sie können beispielsweise gleichzeitig beginnen oder in einer beliebigen Reihenfolge stattfinden. Das Modell wird dann Baseball-Modell genannt. Der Name kommt vom Aussehen der graphischen Darstellung. Mit ein bisschen Fantasie erkennt man einen Baseball, wobei die äußeren Pfeile den Ball und die inneren die Nähte darstellen.

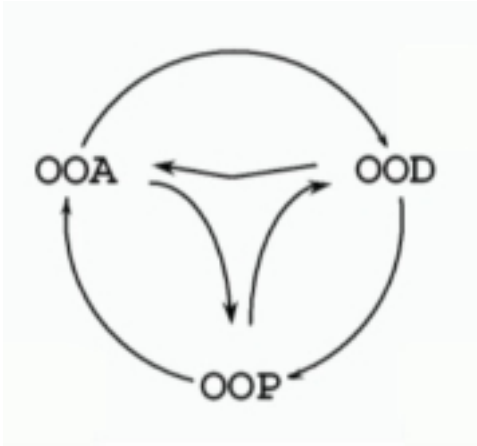


Abbildung 1. Das Baseball-Modell

5.1.2. Interdisziplinäre Teams. Die Entwickler werden in Teams aufgeteilt. Jedes Team bearbeitet einen Teil des Programms, von diesem Teil aber alle Aktivitäten, das heißt OOA, OOD und OOP werden nicht getrennt. Die Verfasser schlagen ein Team vor, das aus folgenden Personen besteht:

- einem bis zwei Experten der realen Welt (das heißt meist die Welt des Kunden)
- zwei Analyseexperten
- einem GUI-Experten
- einem Datenbankspezialist
- einem Experten für Taskmanagement (nur bei Real-Time-Systemen)
- zwei oder drei Programmierern, die die Programmiersprache und die Entwicklungsumgebung beherrschen

5.2. Techniken

Es werden verschiedene Techniken aufgeführt, die in einer Art Fragenkatalog den Entwicklern angeboten werden. Dies beinhaltet Strategien, wo man suchen muss, wie man suchen muss und was berücksichtigt werden sollte. Die Namensgebung ist bei Coad, Yourdon und Nicola etwas anders als bei den meisten Methoden. Deshalb steht der gängige Name immer in Klammern dahinter. Für folgende Aktivitäten werden Techniken bereitgestellt:

- Klassen und Objekte finden
- Strukturen (Beziehungen) erkennen
- Subjekte (Pakete) identifizieren
- Attribute definieren
- Services (Methoden) definieren

Die Techniken können bei OOA und OOD gleichermaßen eingesetzt werden. Um ein paar Techniken kennen zu lernen, wird als Beispiel die Aktivität „Klassen und Objekte finden“ herausgegriffen. Die Anweisungen und

Fragen sind sehr persönlich gehalten, um den Entwickler direkt anzusprechen.

5.2.1. Wo suche ich? Arbeiten Sie mit einem späteren Benutzer eng zusammen. Lernen Sie seine Welt kennen, seine Arbeitsbedingungen, was er gerne bearbeitet, über was er verzweifelt. Lesen Sie alle relevanten Dokumente, lesen Sie betreffende Lexikoneinträge, Artikel und Bücher. Gehen Sie schon abgeschlossene Projekte durch, die für ähnliche Zwecke durchgeführt wurden. Bilden Sie kleine Beispiele, um das Verständnis zu vertiefen.

5.2.2. Was suche ich? Benutzen Sie die Personen-Orte-Dinge-Technik. Alle beim Arbeitsablauf des Endbenutzers vorkommenden anderen Systeme, Geräte, Dinge, Ereignisse, existierende Rollen, Orte und Organisationen müssen auch im Objektmodell vorkommen.

Gehen Sie zuerst in die Breite. Suchen Sie alle möglichen Kandidaten für Klassen und Objekte. Weisen Sie ihnen Beziehungen zu. Reduzieren Sie die Objekte auf die, die wirklich benötigt werden.

Identifizieren Sie Systeme mit ähnlicher Aufgabe. Übertragen Sie mögliche Analogien eins zu eins, oder modifizieren Sie Teile so, dass sie zu Ihrem System passen.

Gehen Sie von einer kleinen Menge an Objekten aus und entwickeln diese in die Tiefe, das heißt recht detailliert. Danach suchen Sie andere Objekte, die von diesen Objekten abhängen.

5.2.3. Was muss ich berücksichtigen? Filterkriterien helfen oft bei der Auswahl der Objekte und Klassen, die das Modell beinhalten soll. Kriterien für Klassen und Objekte sind beispielsweise: Eine Klasse hat ein wohldefiniertes Verhalten. Normalerweise hat eine Klasse mehrere Attribute. Von einer Klasse gibt es mehrere Objekte, beziehungsweise Instanzen.

5.3. Bewertung

Die Methode gibt den Entwicklern Sicherheit, denn sie können sich von den Fragen, beziehungsweise den Anweisungen, leiten lassen. Das Baseball-Modell wirkt sehr gewöhnungsbedürftig. Vor allem die Umstellung, wenn man davor anders gearbeitet hat, ist sehr schwierig. Interdisziplinäre Teams ergeben sich oft auch so von selbst, dies jedoch von vornherein zu steuern scheint sinnvoll zu sein.

6. Nützliches Beiwerk

Dieses Kapitel trägt seinen Namen, weil es Methoden vorstellt, die sehr einfach mit anderen Methoden kombiniert werden können oder auch in verschiedenen Phasen eingesetzt werden können. Sie überdecken eigentlich kei-

ne komplette Phase, wie zum Beispiel den Entwurf, sondern sind eher Hilfsmittel, um zu einem Zwischenergebnis zu kommen oder einen bestimmten Aspekt hervorzuheben.

6.1. CCM (Class Centered Modeling)

CCM wurde hauptsächlich für die Wiederverwendung von Klassen entwickelt. Es stellt eine Notation zur Darstellung einer Klasse zur Verfügung, die es erlaubt, schnell die gewünschten Daten zu finden. Dabei werden zum Beispiel auch geerbte Methoden und Attribute berücksichtigt. Ziel ist es, ähnlich wie in einem guten Klassen-Browser, eine Bibliothek aller Klassen, beispielsweise eines Unternehmens, oder einer Anwendung und Unterstützung bei bestimmten Suchvorgängen bereitzustellen. Im Prinzip stellt CCM nur eine Notation dar.

6.1.1. Die Notation. Hier wird, wie in den meisten Notationen, mit Diagrammen gearbeitet. Neu dabei ist, dass pro Diagramm nur eine einzige Klasse beschrieben wird. Es gibt zwei verschiedene Diagrammarten:

Das „**Class and Information Diagram**“ zeigt die Attribute, sowie Assoziationen und Ober- beziehungsweise Unterklassen. Die folgende Abbildung zeigt nur einige ausgewählte Elemente.

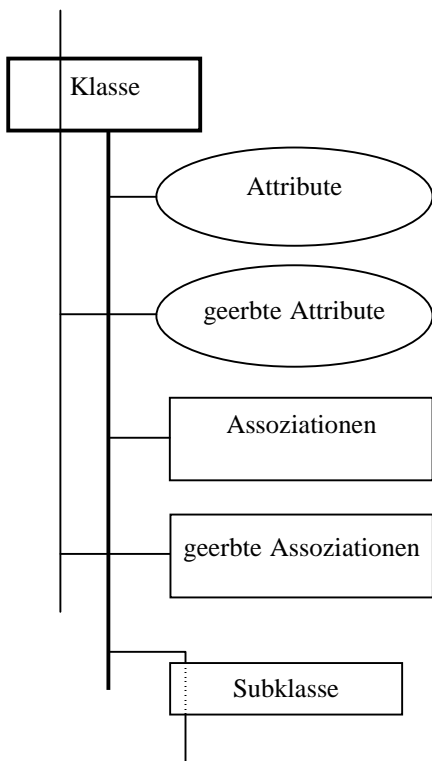


Abbildung 2. Class and Information Diagram

Das „**Class and Events Diagram**“ beinhaltet alle Events, die relevant für die Klasse sind. Diese Art kann noch in zwei Stufen auf das „Class, Events, Operations, and Methods Diagram“ erweitert werden. Dann werden den Events Methoden zugeordnet, die aufgerufen werden, um eine bestimmte Operation (hier ähnlich: Use-Case) auszuführen. Die folgende Abbildung zeigt als kleinen Ausschnitt eines Diagramms die Teilung eines eingehenden Events in zwei weiterführende und die Behandlung eines Request Events.

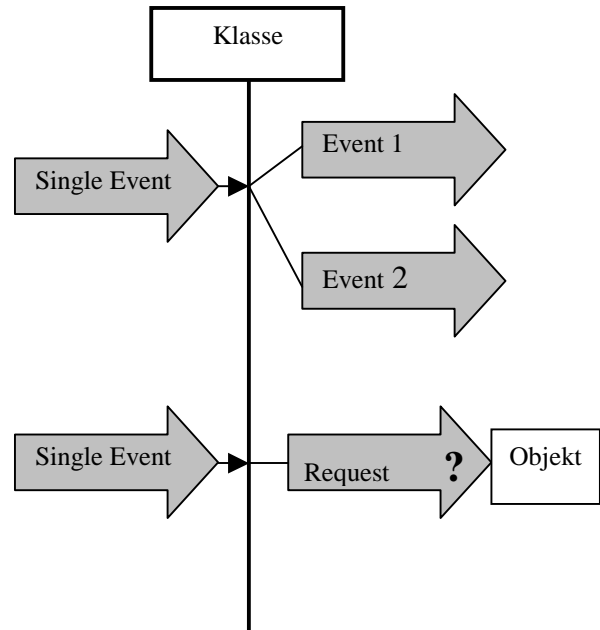


Abbildung 3. Class and Events Diagram

6.1.2. Bewertung. Der Ansatz ist sehr begrüßenswert im Bezug auf die Wiederverwendung. Ein Entwickler kann in einem Diagramm alle relevanten Funktionen zu einer Klasse finden. Der Nachteil ist, dass diese Art der Informationsbereitstellung in hohem Maße redundant ist. Das heißt, die selben Informationen kommen in der Klassenbibliothek an mehreren Stellen vor. Ändert man nun eine Stelle, sollten die anderen auch geändert werden. Von Hand ist das ziemlich mühsam. Deshalb wäre es sehr sinnvoll, die Bibliothek rechnergestützt aufzubauen und solche Fälle automatisch zu bearbeiten. Vorstellbar wäre auch die Zusammenführung von CCM und einem Klassen-Browser, sodass der Klassen-Browser durch CCM um eine graphische Darstellung der Daten erweitert wird.

6.2. CRC-Karten

CRC bedeutet Class-Responsibility-Collaboration und ist ein Verfahren zur Findung von Klassen und zur Erstellung von Szenarien.

6.2.1. Die Sitzung. Alle beteiligten Entwickler versammeln sich an einem Tisch, auf dem leere CRC-Karten bereit liegen. Die Klassen werden nach und nach identifiziert und durch jeweils eine doppelseitige Karte repräsentiert.

Class Name:	
Superclass :	
Subclasses:	
Responsibilities :	Collaboration :

Abbildung 4. CRC-Karte - Vorderseite

Die Karte beschreibt nur die Haupteigenschaften einer Klasse in wenigen Sätzen. Die Karten sind aufgeteilt in zwei Bereiche. Im Responsibility-Bereich wird die Verantwortlichkeit der Klasse beschrieben. Das ist hier gleichbedeutend mit dem grundlegenden Zweck der Klasse, das heißt wofür man diese Klasse angelegt hat. Die Collaboration-Spalte wird mit Klassen gefüllt, mit denen die Klasse zusammenarbeiten muss, um die jeweilige Verantwortlichkeit zu erfüllen.

Attributes:	
Description :	

Abbildung 5. CRC-Karte - Rückseite

Auf der Rückseite der Karte können noch Attribute und weitere Beschreibungen Platz finden. Man kann nun die Vorderseite der Karte als „public“-Teil einer Klasse betrachten und die Rückseite als interne Implementierung, als „private“-Teil.

Nun kann man zum Beispiel einem Entwickler ein paar Klassen zuordnen und Szenarien durchexerzieren. Jeder „spielt“ dabei die ihm zugeordneten Klassen und achtet auf deren Verantwortlichkeiten.

6.2.2. Bewertung.

Durch die Karten entsteht eine Art animierte Diskussion, die sehr vorteilhaft für das Verständnis der eher abstrakten Objektorientierung ist. Außerdem sorgt die Begrenztheit

der Karte für die nötige Kompaktheit der Beschreibung und damit für ein gesundes Abstraktionsniveau. Weiter hilft das Denken in Verantwortlichkeiten allen Beteiligten, das grundlegende, essentielle Verhalten des Systems zu verstehen.

Nachteil der Methode ist allerdings, dass die Sache bei größeren Systemen schnell unübersichtlich wird. Man sollte sich dann wirklich nur auf Hauptklassen beschränken.

6.3. Hauptwortansatz (Noun Approach)

Dieses Verfahren hilft den Entwicklern Klassen zu identifizieren. Der Hauptwortansatz geht von einer Beschreibung des Problems, das die Software lösen soll, in natürlicher Sprache aus. Am besten geeignet ist ein Fachtext oder ein Lexikontext. Ist kein solcher Text vorhanden, muss er zuerst erstellt werden. Nun werden alle Hauptwörter im Text markiert und in eine Liste geschrieben. Danach bildet man anhand der gesammelten Wörter Klassen. Wie das genau funktioniert, wird an einem kleinen Beispiel gezeigt.

6.3.1. Beispiel. Die Stadtverwaltung einer Großstadt will die Müllabfuhr der Stadt rechnergestützt organisieren. Dazu werden Entsorgungsteams gebildet, die aus 3 bis 4 Mitarbeitern bestehen. Jedem Team ist ein Entsorgungsfahrzeug fest zugeordnet. Ein Team ist für die Müllabfuhr in einem oder mehreren Bezirken zuständig. Die genaue Anzahl der von einem Team zu versorgenden Bezirke ist von der Größe des Bezirks abhängig. Es gibt Bezirke der Größe 1, 2 oder 3. Die Summe dieser Größenfaktoren darf pro Team nicht größer als 6 werden, ein Team kann also z. B. 3 Bezirke der Größe 2 entsorgen.

Die gesammelten Hauptwörter:

Anzahl	Großstadt
Bezirke	Mitarbeitern
Bezirken	Müllabfuhr
Bezirks	Stadt
Entsorgungsfahrzeug	Stadtverwaltung
Entsorgungsteams	Summe
Größe	Team
Größenfaktoren	

6.3.2. Bildung der Klassen. Grundsätzlich sind alle Hauptwörter gute Klassenkandidaten. Es gibt jedoch noch Probleme. Redundante Begriffe, beziehungsweise Synonyme, müssen entfernt werden. Im Beispiel meinen die Begriffe „Entsorgungsteams“ und „Team“ genau das gleiche. Einer muss also gestrichen werden. Es gibt noch eine Reihe von Regeln, die der Entwickler jedoch in jedem Fall nachprüfen muss.

Gute Klassenkandidaten sind auch physische Objekte (Mitarbeiter).

Oft bezeichnen Eigennamen Objekte und können zu Klassen Anlass geben, wenn mehrere für das System bedeutende Objekte vorkommen und eine eigenständige Behandlung erfahren.

Abstrakte Objekte (nicht physische Objekte) sind häufig mit physischen Objekten verbunden und stellen lediglich eine Sicht darauf dar (zum Beispiel: Größe oder Team).

Ein Klassenkandidat, der keine Attribute enthält, ist meist ein Attribut einer anderen Klasse. Klassenkandidaten die eigene Attribute besitzen, sind meist Klassen.

Klassenkandidaten, die ein wohldefiniertes Verhalten haben, sind auch meist Klassen (hier: Müllabfuhr).

6.3.3. Bewertung. Die Gefahr dieses Verfahrens ist, dass das Ergebnis oft eine Vielzahl von Klassen ohne Beziehungen beinhaltet. Außerdem birgt die Reduzierung um synonyme und irrelevante Klassen einen sehr hohen Aufwand. Weiter ist ein hohes Fachwissen erforderlich, um Fachtermini zu erkennen und um zu entscheiden, ob ein Hauptwort nun Klasse, Attribut oder Methode ist. Die Vorteile liegen auf der Hand: Zum einen ist das Verfahren sehr einfach anzuwenden und zum anderen universell einsetzbar.

7. Zusammenfassung

In diesem Kapitel wird versucht, auffällige Aspekte vieler Entwurfsmethoden herauszustellen. Eine bedeutende Auffälligkeit ist, dass die Prozesse sehr ähnliche Schritte oder Aktivitäten beinhalten. Diesem Punkt, der die Ähnlichkeit vieler Methoden herausstellt, ist ein eigenes Kapitel (7.2) gewidmet. Am Schluss findet sich noch ein Ausblick, beziehungsweise ein Fazit.

7.1. Auffälligkeiten

7.1.1. Herkunft. In ein paar Fällen sieht man den Methoden schon genau an, aus welcher Ecke ihre Väter kommen. Es gibt zum Beispiel Methoden, die eine Software als Behörde abbilden und auf dieser Grundlage Szenarien erstellen.

Oder zum Beispiel OOIE (Object-Oriented Information Engineering), das eindeutig aus der BWL-Richtung kommt. Dabei werden zuerst die strategischen Geschäftsfelder des Kunden erfragt und die Softwarekonzeption danach ausgerichtet.

Die schon vorgestellte SSADM ist stark an die Datenbankentwicklung angelehnt.

7.1.2. Verzahnung. Viele Methoden verweisen in bestimmten Phasen oder Schritten auf andere Methoden, wie zum Beispiel Booch auf die CRC-Karten. Andere Metho-

den bauen dafür auf einer Analyse nach Booch auf. Die Methoden stehen also oft nicht nur für sich, sondern sind mit anderen verzahnt.

7.1.3. Bibliotheken. Bei den meisten Methoden findet sich der Ansatz, dass die gefundenen Informationen konsequent in einer Art Bibliothek abgelegt werden. Zum einen, um erarbeitetes Wissen oder Struktur zu sichern, und zum anderen, damit die Wiederverwendung von Klassen oder Teilsystemen erleichtert wird. Viele Ansätze haben aus diesem Grund auch ein Repository vorgeschlagen oder umgesetzt.

7.1.4. Analyseunterstützung. Fast alle Methoden beschäftigen sich neben dem Entwurf mit der Analyse. Man hat erkannt, dass eine Methode nur schwer beim Entwurf alleine ansetzen kann. Es ist einfacher, schon die Analysephase auf das Verfahren auszurichten.

7.2. Ähnlichkeiten der Prozesse

Es scheint einen Konsens zu geben, dass die Entwurfsphase in zwei große Teile gespalten wird. Sie heißen oft unterschiedlich und der erste wird auch zum Teil der Analyse zugeordnet, aber grundsätzlich stehen sie fest:

- Klassen- und Objektstruktur
- Feinentwurf

7.2.1. Klassen- und Objektstruktur. Ziel hierbei ist es, eine zunächst statische Struktur aufzubauen, die alle Klassen und Objekte enthält, die in der echten Welt (beim Kunden) wichtig sind. Folgende Schritte werden meist angeführt:

- Identifizieren der Klassen und Objekte
- Beziehungen finden zwischen diesen Elementen
- Attribute und Methoden der Klassen festlegen
- Iterieren und Validieren

7.2.2. Feinentwurf. Bei dieser Phase ist das Ziel, die im oberen Schritt entwickelte Klassen- und Objektstruktur auf den Rechner abzubilden. Hier kommen physische Aspekte wie Betriebssystem, Datenverwaltung oder Ausgabegeräte ins Spiel. Es erfordert immer auch neue Klassen und Objekte, die die Kommunikation zwischen der physischen Schicht und der Objektstruktur von oben ermöglichen.

Als zweiter wichtiger Punkt werden hier vermehrt dynamische Sichten eingeführt, das heißt Szenarien oder Use-Cases werden durchgespielt, um Schwachpunkte zu identifizieren und zu beseitigen.

7.3. Ausblick

Es gibt bisher keine Schnittstellenvereinbarung, die die Schnittstellen zwischen Analyse und Entwurf festlegen. Das würde sich wohl auch sehr schwierig gestalten und viele Experten sagen, es ist fast unmöglich, da in der Objektorientierung Analyse und Entwurf sehr eng zusammenhängen und schon fast verschmelzen.

Deshalb und wie auch schon in Kapitel 7.1.4. angesprochen, geht der Trend zur umfassenden Unterstützung des gesamten Softwareentwicklungsprozesses. Ein Beispiel für so eine Methode, beziehungsweise Werkzeug, ist der „Rational Unified Process“ [5].

Zum Abschluss noch ein kleiner Abschnitt aus [3], zuerst in deutsch und danach für diejenigen, die lieber das Original lesen, in englisch.

„Der amateurhafte Softwareingenieur sucht immer nach Magie, nach sensationellen Methoden oder nach Tools, deren Anwendung Softwareentwicklung trivial machen. Den professionellen Softwareingenieur zeichnet aus, dass er weiß, dass solche Allheilmittel nicht existieren.“

“The amateur software engineer is always in search of magic, some sensational method or tool whose application

promises to render software development trivial. It is the mark of the professional software engineer to know that no such panacea exists.”

8. Literatur

[1] Meyers Lexikonredaktion, *Duden - Das neue Lexikon*, Dudenverlag, Mannheim, 1996.

[2] Hutt, Andrew T.F., *Object Analysis and Design : Description of Methods*, John Wiley & Sons, Inc., New York, 1994

[3] Grady, Booch, *Object-oriented analysis and design with applications – 2nd edition*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994

[4] Boehm, Barry, “A Spiral Model of Software Development and Enhancement”, *IEEE Computer*, vol.21, #5, May 1988, pp 61-72.

[5] Kruchten , Philippe, *The Rational Unified Process – An Introduction*, Addison-Wesley, Reading, Massachusetts, 1998