Evaluation der OOSE-Methode mit Hilfe des NIMSAD-Frameworks

Studienarbeit am Fachbereich Informatik, Arbeitsbereich Softwaretechnik der Universität Hamburg, Oktober 98

von Nilgün Özek

Betreuung: Dr. Ralf Klischewski

Nilgün Özek
Martin Niemöllerstr. 13
22880 Wedel
Tel: 04103/97834
Matr.Nr: 4175480

Inhaltsverzeichnis

Einleitung

Methoden sollen Vorgaben für ein systematisches Vorgehen bei der Softwareentwicklung liefern und bei der Problemlösung in dem jeweiligen Anwendungsgebiet den Entwicklern bei den Bemühungen der Problemlösung helfen [11]. Es gibt aber heutzutage eine Fülle von Software-Entwicklungsmethoden, so daß es sehr schwierig ist diejenige Methode auszuwählen, die am besten für die konkrete Situation geeignet ist. Dazu kommt noch, daß viele Methodenentwickler/innen behaupten, daß ihre Methoden universell einsetzbar sind und sie ungern die Schwachpunkte ihrer Methoden zugeben, um eine weite Benutzung ihrer Methoden zu ermöglichen [15].

Die Auswahl und die Anpassung der verwendeten Methode an die jeweilige Situation ist eine sehr wichtige und zugleich auch eine schwierige Aufgabe bei der Softwareentwicklung. Die Wichtigkeit der Auswahl von Methoden wird z.B. von Christiane Floyd folgendermaßen zum Ausdruck gebracht, indem sie sagt, daß es keine richtige Methode gibt und daß die Entwickler/innen vielmehr die für sie relevanten Methoden auswählen und anpassen sollen [10]. Hierdurch ergibt sich aber die Frage, wie die Entwickler/innen die relevante Methode auswählen und verstehen sollen, um auch die geeigneten Anpassungen an der Methode durchführen zu können. Die Softwareentwickler/innen brauchen bei der Auswahl, beim Verstehen einer Methode Unterstützung, wie z.B. NIMSAD-Framework.

NIMSAD steht für **N**ormative **I**nformation **M**odel-based **S**ystem **A**nalysis und **D**esign. Es ist ein methodenunabhängiges Rahmenwerk, durch das alle Methoden laut Nimal Jayaratna verstanden und beurteilt werden können. Dieses Rahmenwerk soll nach Jayaratna den Softwareentwickler/innen bei der Auswahl und dem Einsatz einer geeigneten Methode unterstützen.

Seit einigen Jahren gilt der Begriff „Objektorientierung“ als ein Schlagwort, womit sich viele Universitäten, Firmen und andere Organisationen beschäftigen. Es wurden aus diesem Grund auch viele objektorientierte Methoden entstanden, die sich durch ihre Konzepte unterscheiden. In dieser Studienarbeit wird von diesen Methoden die OOSE-Methode von Jacobson et.al. zur Bewertung nach NIMSAD-Framework ausgesucht, weil sie durch ihren Use-Case-Konzept und ihren Anspruch die Softwareentwicklung zu industrialisieren und rationalisieren in Praxis und Wissenschaft in letzter Zeit im Blickpunkt geraten ist. Es werden in dieser Arbeit die Konzepte und die Sichtweise der OOSE bei der Softwareentwicklung untersucht, mit dem Ziel die Methode zu verstehen und sie dann nach NIMSAD-Framework kritisch zu evaluieren.

Zu der Methode OOSE existieren viele Veröffentlichungen. Das Use-Case-Konzept hat insbesondere durch die Benutzung im UML (Unified Manipulation Language) einen weiteren Bekanntheitsgrad erreicht. In dieser Studienarbeit wird als Grundlage das im Jahre 1992 publizierte Buch „Object-Oriented Software Engineering“ von Jacobson et.al. benutzt.

Die Literaturresearche über das NIMSAD-Framework ergab, daß lediglich das Buch „Understanding and Evaluating Methodologies“ von Jayaratna veröffentlich wurde und somit grundlegend für diese Arbeit war.

Ziele dieser Studienarbeit sind:

1. Die OOSE (Object-Oriented Software Engineering) Methode von Ivar Jacobson durch die Anwendung des NIMSAD-Framework zu evaluieren und dabei zu untersuchen, ob das NIMSAD-Framework auch für die Evaluation der OOSE-Methode von Jacobson et.al. herangezogen werden kann.

2. Schlußfolgerungen aus der Evaluation der OOSE Methode ziehen.

Im *ersten Kapitel* wird, auf die Definitionen der Begriffe „Software- Engineering“ und „Methode, auf die strukturierte- und objektorientierte Methoden und anschließend auf die Probleme in der Softwareentwicklung in Verbindung mit Methoden eingegangen.

Die Beschreibung der Methode OOSE (Object-Oriented Software Engineering) erfolgt dann im *zweiten Kapitel*. Hier werden die Hintergrundtechnologien der OOSE-Methode, Systementwicklung nach Jacobson et.al., die allgemeinen Vorgehensweise und die Modelle dieser Methode beschrieben. Zum Schluß werden die methodischen- und Managementaspekte der Methode gegenübergestellt.

Die NIMSAD-Framework wird im *dritten Kapitel* dargestellt. Zunächst wird der Unterschied zwischen einer Methode und einem Framework aus der Sicht von Jayaratna und die Anwendung des Frameworks bei der Evaluation von Methoden beschrieben. Anschließend werden die Elemente des NIMSAD-Frameworks und die Schritte erklärt.

Der Gegenstand von *Kapitel vier* und der Hauptbestandteil dieser Studienarbeit ist die Beurteilung der OOSE-Methode durch die Anwendung des NIMSAD-Framework. Es wird hier untersucht, inwieweit die OOSE-Methode die Elemente des Frameworks berücksichtigt und wie die Schritte des Frameworks in der Methode angeordnet sind. Dabei wird der Fragenkatolog, der im Kapitel 3 dargestellt ist, als Unterstützung für die Evaluation benutzt.

Das *fünfte Kapitel* enthält die Ergebnisse dieser Studienarbeit. Zum einen wird die Bewertung des NIMSAD-Frameworks im Hinblick ihre Eignung für die Evaluation der OOSE-Methode und zum anderen die Schlußfolgerungen aus der Bewertung der OOSE-Methode mit NIMSAD-Framework, dargestellt.

1. Software-Entwicklung und Entwicklungsmethoden

1.1. Der Begriff „Software-Engineering

Am Anfang dieser Studienarbeit wird kurz auf die Entstehung, die Definition und die Leitbilder bei der Software-Engineering eingegangen und anschließend werden Probleme der Software-Engineering in Verbindung mit Methoden diskutiert.

Der Begriff „Software-Engineering“ entstand aufgrund der Softwarekrise in den 60er Jahren. Die Softwarekrise wurde durch Mangel an Methoden und Ausbildung der Softwareentwickler, zunehmende Komplexität der zu verwendeten Hilfsmittel (Sprachen, Betriebssysteme, Rechner) und der zu bearbeitenden Probleme sowie fehlende Einsichten in der Arbeitsorganisation und Kommunikation ausgelöst. Aus diesen Gründen fanden im Jahre 1968 in Garmisch zum Thema „Software Engineering“ und 1969 in Rom zum Thema „Software Engineering Techniques“ internationale Konferenzen statt. Auf diesen Konferenzen entstand die Forderung, daß Software den Charakter einer Ingenieurdisziplen aufweisen sollte [9].

Inzwischen existieren viele Definitionen des Begriffs „Software-Engineering. Es gibt aber bis heute keine allgemein anerkannte Definition für diesen Begriff [21]. Aus diesem Grund werden nachfolgend einige Definitionen aus der Literatur zitiert, um Gemeinsamkeiten zu finden.

Floyd definiert in [9, S.1] Software Engineering folgendermaßen:

> Unter Software Engineering versteht man theoretische Grundlagen sowie praktische Verfahren zur ingenieurmäßigen Herstellung von Software. Softwaretechnik befaßt sich mit theoretischen Grundlagen, methodische Anleitung, technischer Unterstützung sowie mit der Organisation von Projekten zur Softwareentwicklung.

Balzert schreibt in [1, S.36]

> Softwaretechnik ist die zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden, Konzepten, Notationen und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Software-Systemen. Zielorientiert bedeutet die Berücksichtigung z.B. von Kosten, Zeit, Qualität.

Der gemeinsame Bestandteil dieser Definitionen liegt in der ingenieurmäßigen Herstellung der Software. Das Wort „ingenieurmäßig“ wird im Zusammenhang mit der Software-Engineering in der Literatur oft diskutiert und erfordert eine genaue Erklärung [20, 18]. An dieser Stelle ist zu erwähnen, daß Christiane Floyd aus diesem Grund in der Software-Engineering zwischen zwei Sichtweisen[1] unterscheidet und einen Wandel zur Design Sicht vorschlägt, die einerseits die Einbettung von Software in menschliche Zusammenhänge berücksichtigt und andererseits technische, wirtschaftliche, soziale und ästhetische Gesichtspunkte integriert [9].

[1] Vergleiche hierzu Abschnitt 1.2. in diesem Kapitel.

1.2. Leitbilder bei der Softwareentwicklung

Leitbilder sind für die Softwareentwicklung und für den Entwicklungsprozeß von sehr großer Bedeutung. Nach Floyd wird in der Software-Entwicklung ein Leitbild als die jeweils vorherrschende Sichtweise für die Gestaltung von Softwaresystemen charakterisiert. Somit bestimmt das Leitbild, wie der jeweils betrachtete Ausschnitt von Realität wahrgenommen, verstanden und gestaltet wird [11].

Im Zusammenhang mit Methoden bedeutet dies, daß sie die Sichtweisen ihres Entwicklers für die Softwareentwicklung wiedergeben und damit den Software-Entwicklungsprozeß wesentlich beeinflussen. Nimal Jayaratna schreibt in diesem Zusammenhang:

> „Methodologies reflect the Weltanschauungen of their creators. The steps of methodologies, their structure, chosen models and implied values give an indication as to how their creator perceives reality." [17, S.42]

An dieser Stelle möchte ich auf die zwei Sichtweisen, die schon vorher erwähnt worden sind, bei der Softwareentwicklung nach Christiane Floyd eingehen. Floyd unterscheidet zwischen zwei Sichtweisen (Perspektiven) in der Softwarentwicklung: *Produktionssicht* und *Design-Sicht* [9,10].

Sie verwendet den Begriff „Produktionssicht", um die idealisierten Grundannahmen des Software-Engineering über die Konstruktion von Software zusammenzufassen.Bei der Produktionssicht wird Software als eigenständiges Produkt aus eigenständigen Programmen und Dokumentationen verstanden. Die Softwareentwicklung hingegen wird als Produktion im Sinne einer Fertigung betrachtet. Dabei wird die Metapher von der Softwarefabrik verwendet. Die Metapher von Softwarefabrik wird auch von Enders kritisiert. Er sagt dazu: „ Es wird damit suggestiert, daß Software-Entwicklung eine geistlose Tätigkeit ist, die man am meisten von Arbeitern im blauen Anzug erledigen läßt (...) Daß sie damit dem Bild des in der Praxis arbeitenden Software-Entwicklers Schaden zufügen, ist Ihnen wohl nicht bewußt." [6]

Bei der von Floyd bevorzugten Design-Sicht steht die Verständigung zwischen Beteiligten und die Einbettung von Software in einem veränderlichen Einsatzkontext im Vordergrund. Das Design betrifft sowohl das Produkt Software, seinen Einsatz als auch den Entwicklungsprozeß und seine methodische Unterstützung. Im Gegensatz der Produktionssicht wird die Softwareentwicklung hierbei nicht menschenunabhängig gesehen.

Floyd leitet aus der Design-Sicht Anforderungen wie Partizipation, Prozeßorientierung, evolutionäres Vorgehen für Theoriebildung und Methodenentwicklung ab. In der Literatur werden diese Begriffe, insbesondere Partizipation und evolutionäres Vorgehen viel diskutiert. Es wird von den meisten wissenschaftlichen Autoren gefordert, daß bei der Softwareentwicklung die Anwender an den Prozeß beteiligt werden und die Softwareentwicklung evolutionär vorgehen sollte [13, 9].

1.3. Der Begriff „Methode“

Bevor die Methoden in der Softwareentwicklung näher betrachtet werden, wird zunächst der Begriff „Methode“ erläutert.

Der Begriff „Methode“ wird aus dem Griechischen methodos (das Nachgehen, Verfolgen) abgeleitet und wird als „ein planmäßiges Verfahren zur Erreichung eines bestimmten Zieles“ [24, S.454] oder mit „Erkenntnisweg, planmäßiges Vorgehen zum Erlangung oder Begründung von Wissen“ [14,s.390]] übersetzt. Im Duden Informatik wird ausführlich auf dem Methodenbegriff eingegangen und der Methodenbegriff in der Informatik folgendermaßen definiert:

„Wörtliche Bedeutung : Der Weg zu etwas. Bedeutung in der Informatik: systematische zielgerichtete Vorgehensweise, sowie planmäßiges Verfahren, welches für eine Vielzahl von Problemen zu einer sinnvollen Lösung führt. (...) Jede Wissenschaft besitzt eigene Methoden, und die Menge der Methoden legt in der Regel das Selbstverständnis einer Wissenschaft fest. In diesem Sinne ist die Informatik die Methodenlehre für den Bereich der Information.“ [5, 404ff]

An dieser Stelle werden noch andere Definitionen für den Begriff Methode aus der Literatur zitiert, um Gemeinsamkeiten bzw. die Unterschiede zu zeigen.

Stein schreibt in [25, S.23]:

> Eine Softwareentwicklungsmethode ist eine Methode, die zur Entwicklung von Softwaresystemen eingesetzt wird. Sie beinhaltet neben der reinen Vorgehensweise in der Regel eine spezielle, textuelle oder graphische Notation zur Dokumentation des Softwaresystems.

Bei Balzert [1, S.36] wird sie folgendermaßen definiert:

> Methoden sind planmäßig angewandte, begründete Vorgehensweisen zur Erreichung festgelegten Zielen. Sie geben an, welche Konzepte wie und wann verwendet werden, um die festgelegten Ziele zu erreichen.

Floyd definiert Methode in [11, S.650]:

> Methoden liefern Vorgaben für ein systematisches Vorgehen bei der Softwareentwicklung. Der Begriff „Methode“ bezieht sich sowohl auf einzelne Aktivitäten wie Analyse, Entwurf oder Programmierung als auch auf die Softwareentwicklung insgesamt. Sie verkörpern eine Sicht der Softwareentwicklung (Perspektive), beziehen sich auf einen Anwendungsbereich und geben Richtlinien in Form von Techniken, Werkzeugen und Organisationsformen.

All diese Definitionen beschreiben eine Methode als systematische und planmäßige Vorgehensweise bei der Softwareentwicklung. Floyd betont außerdem, daß die Methoden auch eine Sicht der Softwareentwicklung verkörpern. Aber in keiner dieser Definitionen wird auf die Subjektivität der Methodenanwender/innen eingegangen. Im Kapitel 3 dieser Studienarbeit wird im Rahmen des NIMSAD-Framework darauf eingegangen.

1.3.1. Methoden in der Softwareentwicklung

Die Softwareentwicklungsmethoden, die in der Praxis und Forschung relevant sind, orientieren sich an zwei unterschiedlichen Paradigmen. Sie können demnach in zwei Methodengruppen klassifiziert werden: Strukturierte Methoden und objektorientierte Methoden [25] Die Unterscheidung zwischen dieser Methodengruppen ist auch wichtig, um die große Menge der Methoden in der Praxis zu ordnen [11].

Strukturierte Methoden sind in der Praxis sehr weit verbreitet und es existieren eine große Menge CASE-Tools zur Unterstützung für diese Methoden. Die Entwicklung dieser Methoden ist in der Forschung weitgehend abgeschlossen. Die objektorientierten Methoden hingegen finden in der Praxis erst in den letzten Jahren eine weite Verbreitung. Das Interesse in der Forschung gilt aufgrund der Verbreitung der objektorientierten Sprachen mehr den objektorientierten Methoden [25].

In den folgenden Abschnitten werden die strukturierten- und objektorientierten Methoden näher beschrieben, um die Unterschiede zu zeigen.

1.3.1.1. Strukturierte Methoden

Strukturierte Methoden sind als die erste Generation von Methoden zur Softwareentwicklung von großer Bedeutung. Die ersten strukturierten Methoden entstanden Anfang der 70er Jahre ausgehend von der strukturierten Programmierung. Am verbreitetsten sind die SA (Structered Analysis) von Tom DeMarco verbunden mit SD (Structured Design) von Yourdon. Struktured Analysis wurde von verschiedene Autoren im Laufe der Zeit modifiziert und weiterentwickelt. Die wichtigsten SA-Varianten sind: *Structered Analysis* von Weinberg (1978) *Structered Systems Analysis* von Gane /Sarson (1979), *Essential Systems Analysis* von McMenamin/Palmer (1984) und *Modern Structered Analysis* von Yourdon (1989) [1].

Die strukturierten Methoden stellen die ablaufbezogene Funktionsmodellierung für eine Anwendung in den Vordergrund. Es wird in der Praxis unabhängig davon ein globales Datenmodell erstellt. Anschließend müssen Funktionsmodell und Datenmodell in einem eigenen Arbeitsschritt aufeinander abgestimmt werden. Diese Methoden streben ein standardisiertes Vorgehen nach dem Top-Down-Prinzip an, so daß von einem umfassenden, abstrakten Modell zu einer konkreten Detaillierung und programmiersprachlichen Realisierung übergegangen werden kann. Bei allen strukturierten Methoden

stellt der Übergang zwischen Modellebenen (z.B. Analyse, Entwurf, Implementierung) ein großes Problem dar [11].

1.3.1.2. Objektorientierte Methoden

Der Begriff „objektorientiert“ ist in den letzten Jahren eines der meist benutzten Schlagworte in der Softwareentwicklung. Es war bereits 1967 mit Simula die erste objektorientierte Programmiersprache verfügbar, jedoch erst mit der Verbreitung von Smalltalk nach 1980 begann die Entwicklung der objektorientierten Methoden [11] Die erste objektorientiert klassifizierte Methode war die im Jahre 1989 publizierte *Object-Oriented Requirements Specification Method* (OOS) von Sidney C. Bailin [25].

Seit Beginn der 90er Jahre nehmen die Anzahl der objektorientierten Methoden stark zu. Heute gibt es eine Vielzahl von objektorientierten Methoden, so daß es kaum möglich ist, sie in ihrer Vielfalt zu überschauen. Wichtige Vertreter sind die OMT (*Object Modeling Technique*) von Rumbaugh, OOD (*Object-Oriented Design*) von Booch und die OOSE (*Object-Oriented Software Engineering*) von Jacobson. Eine ausführliche Darstellung, Vergleich und Bewertung befindet sich in [25].

Die klassische Softwareentwicklung unterscheidet zwischen Analyse und Entwurf. Die Unterscheidung zwischen Analyse- und Entwurfsmethoden ist für die Auswahl der Methoden wichtig .Strukturierten Methoden trennen strikt zwischen Analyse und Entwurf. Die Struktur des Analysemodells ist in der Regel nicht in dem Entwurfsmodell wiederzufinden. Beide Modelle benutzen andere Modellierungskomponenten. Im Gegensatz zu strukturierten Methoden gibt es in den objektorientierten Methoden eine viel weniger scharfe Trennlinie zwischen diesen beiden Phasen. Objektorientierte Analyse und objektorientierte Entwurf bauen auf den selben Modellierungskomponenten (Instanzen, Attribute, Operationen und Klasen) auf. Deswegen ist die Abgrenzung zwischen objektorientierter Analyse und objektorientierter Entwurf schwieriger als bei den strukturierten Methoden. Es wird auch in der Literatur der Übergang zwischen diesen Phase unterschiedlich interpretiert. Analysschritte mancher Methoden sind in anderen beispielsweise bereits Teil des Entwurf und umgekehrt.

Nach Stein [25] sind die wesentliche Unterscheidung zwischen der objektorientierten Analyse und dem objektorientierten Entwurf nicht die Modellierungskomponenten sondern die Bereiche, aus denen die modellierten Objekte und Klassen stammen. Er unterscheidet zwischen dem *Problembereich* (Analyse) und dem *Lösungsbereich* (Entwurf). Die Objekte und Klassen der Problembereiches werden in der Analyse identifiziert und spezifiziert. Die Objekte und Klassen des Problembereiches zusammen mit den Objekten und Klassen zur Realisierung der Benutzungsoberfläche, zur Verwaltung der Datenmengen und zur Prozeßverwaltung beinhaltet der Lösungsbereich.

1.4. Probleme bei der Software-Engineering

Wie schon am Anfang dieser Kapitel erwähnt, sprach man Ende der 60er Jahre von einer Softwarekrise. Die Software-Engineering hat zwar für Teilbereiche der Softwareentwicklung, besonders für Modellierung, Softwareentwurf und Programmierung große Fortschritte gebracht. Die Software-Engineering steht heute aber immer noch vor Problemen, besonders „bei den kreativen und kooperativen Anteilen des Konstruktionsprozesses, die einer Formalisierung nicht zugänglich sind und bei organisationsbezogener Softwareentwicklung, wo Probleme nicht fest vorgegeben, Anforderungen veränderlich und Softwareprodukte im Einsatz eng mit Arbeits- und Kommunikationsprozessen von Einzelpersonen oder von Gruppen verzahnt sind.“ [10, S.29]

Aus diesen Gründen beschäftigen sich im Bereich der Software-Engineering die Wissenschaft und viele andere Organisationen seit mehreren Jahren mit der Verbesserung der Softwareerstellung. Zu diesem Zweck wurden Vorgehensweisen (Wasserfallmodell, Spiralmodell usw.) vorgeschlagen und viele Methoden entwickelt. Viele dieser Methoden betonen nur die technische Seite des Software-Entwicklungsprozesses und vernachlässigen der am Prozeß beteiligten Personen.

Ein Hauptproblem dabei ist, daß die Software-Engineering und damit auch die Methodenentwicklung in Wissenschaft und Wirtschaft in ihren theoretischen und praktischen Problemstellungen, ihren Interessen und ihrer Sprache auseinanderliegen [4]. Denert bringt das Verhältnis zwischen Wissenschaft und Praxis in [4, S. 297f.]mit folgenden Sätzen zum Ausdruck:

> „Wissenschaftler neigen dazu, Probleme für ihre geliebten Theorien zu suchen, statt neue Lösungen für gegebene Aufgaben (...) Praktiker wollen Richtlinien, Verfahrensregeln, Faustformeln, möglichst einfach, griffig, werkzeugunterstützt (...) Das Argument, eine Methode sei theoretisch begründet, ist ihnen eher suspekt, praktische Bewährung alles, auch wenn dahinter nur konzeptionslose Bastelei steckt“

Diese Sätze zeigen, auch wenn sie provakativ sind, den Zwiespalt der Wissenschaft und Praxis. Für einen erfolgreiche Softwareentwicklung müssen sich beide Seiten näherkommen. Die Wissenschaft sollte vielmehr versuchen die Informatiker/innen ermutigen, „praktische Probleme mit theoretisch fundierten Methoden kreativ zu lösen.“ [ebd. 298]. Christiane Floyd erkennt dieses Problem und formuliert dies in [7, S.272] folgendermaßen:

> „ Wir sollen anstreben, auf das Vorwissen unserer Anwender organisch aufzusetzen, anstatt sie mit unserer Begriffswelt zu überrollen (...) Wir sollten Methoden erproben, bevor wir sie verkaufen (...) Letztlich gibt es keine Methoden, sondern Menschen als Träger von Methoden. „

Eine Untersuchung zum Methodeneinsatz im Softwareentwicklungsprojekten bekräftigt dieses Verhältnis zwischen Praxis und Wissenschaft. Das Ergebnis der Untersuchung ist: „Hinsichtlich des

Werkzeugs- und Methodeneinsatzes scheint noch eine recht große Lücke zu sein, zwischen theoretisch erarbeitenden Lösungen und dem, was in der Praxis verwendet wird, zu bestehen.“ [3, S.58]

Ich möchte dieses Kapitel mit den Sätzen von Floyd schliessen. Die in der Praxis nicht genügend umgesetzten der erlernten Software-Engineering-Wissen, Akzeptanzprobleme für Methoden und die Mehrarbeit, die durch Methoden und Tools entstehen, könnten nach Floyd bei der Softwareentwicklung zu einer neuen Softwarekrise der 90er Jahre führen.[10, S.31]

Aus diesen Gründen sollte die Evaluation von Methoden eine wichtige Rolle zwischen Praxis und Wissenschaft spielen. Die Wissenschaft kann aus der Evaluation des Methodeneinsatzes in Praxis die Lücken der Methoden herausarbeiten, mit dem Ziel zum einen ihre Methoden zu verbessern und zum anderen einen bessere Akzeptanz und Einsatz der Methoden in der Praxis zu erreichen.

Nachdem in diesem Kapitel die Grundlagen und die Problembereiche in Verbindung mit Methoden erläutert wurden, soll im folgenden Kapitel die evaluierende Methode OOSE von Jacobson et.al. dargestellt werden.

2. Object-Oriented Software Engineering(OOSE) von Jacobson et al.

Die Methode Object-Oriented Software Engineering (OOSE) wurde von Ivar Jacobson, Magnus Christerson, Patrik Jonsson und Gunnar Övergaard bei Objective Systems in Schweden entwickelt und 1992 als Buch veröffentlicht. Sie wird durch das Werkzeug Objectory unterstützt [15].

Die Entwickler der Methode OOSE haben den Anspruch aus dem Software-Entwicklungsprozeß einen rationalen industriellen Prozeß zu machen und vergleichen aus diesem Grund die Softwareindustrie mit der Bauindustrie. Die Methode OOSE betrachtet die Systementwicklung als eine industrieller Aktivität und somit benötigt sie die gleichen Anforderungen wie in der Industrie. Sie gehen dabei davon aus, daß die Ergebnisse des Software-Entwicklungsprozesses unabhängig von den beteiligten Individuen sind. Die Entwickler der Methode sind davon überzeugt, daß die Benutzung der OOSE zu besseren Systemen führen wird.

OOSE wurde in der Telekommumikationsindustrie entwickelt und kann nach Jacobson et.al sowohl in diesem Bereich als auch in administrativen und technischen Bereichen eingesetzt werden. Für sie ist die Systementwicklung ein Prozeß kontinuierlicher Veränderung. Sie betonen dabei die inkrementelle Vorgehensweise und Wiederverwendbarkeit bei der Systementwicklung.

Die Entwickler der Methode OOSE empfehlen Prototyping, um Software-Applikationen besser zu verstehen. Sie bevorzugen dabei weiterverwendbare Prototypen gegenüber dem sogenannten rapid prototyping, und sagen: „Rapid prototyping is in relatively bad repute today; there are even those who classify it as ´quick and dirty´“ [ebd. S. 25].

OOSE zeichnet sich gegenüber anderen objektorientierten Methoden durch eine anwendungsorientierte Vorgehensweise (use case driven approach) aus. Durch die Use Cases soll der Ablauf einer typischen Benutzung des Systems beschrieben werden. Das Use Case Modell ist das zentrale Modell, aus dem die anderen Modelle abgeleitet werden (siehe 2.4.1)

Aufgrund der vielen Veröffentlichungen und Berücksichtigung dieser Methode bei Methodenvergleichen[2], kann man davon ausgehen, daß die Methode OOSE sowohl in der Wissenschaft als auch in der Praxis aufgrund ihres Use-Case-Konzept und ihrer Interaktionsdiagramme eine weite Verbreitung findet. Jacobson et.al. geben dazu konkrete Zahlen und sagen, daß die OOSE im Jahr 1993 in mehr als 20 Projekten von verschiedener Größe (3-50 Mannjahre) eingesetzt wurde. Zur Zeit wird es in mehr als 50 Projekten, vor allem in Europa und Nordamerika eingesetzt.

[2] OOSE wurde in vier Methodenvergleichen berücksichtigt. (Stand 1994) (vgl. Stein S.30)

Im weiteren dieses Kapitels werden die Hintergrundtechnologien, die Vorgehensweise und die Modelle der OOSE dargestellt.

2.1. Hintergrundtechnologien

Die Methode Object-Oriented Software Engineering (OOSE) kombiniert drei Techniken, die nachfolgend aufgeführt werden. Diese sind:

1. **Objektorientierte Programmierung:** Sie wurde in den 60er Jahren entwickelt. Sie fand erst in jüngster Zeit eine weite Verbreitung. Die OOSE nutzt von den objektorientierten Konzepten hauptsächlich die Vererbung, Datenkapselung, Beziehungen zwischen Klassen und Objekten.

2. **Konzeptuelle Modellierung:** Sie wurde in den 70er Jahren in verschiedenen Kontexten benutzt. Das Ziel hierbei ist, Modelle von dem zu analysierenden System oder Organisation zu kreieren. In der OOSE wird diese Technik zum einen um die objektorientierten Konzepte und zum anderen um die Fähigkeit des dynamischen Verhaltens auszudrücken, erweitert. Die Modelle, die im Rahmen der OOSE entwickelt werden, sollen dazu dienen, das System zu verstehen und eine gute Systemarchitektur zu erzielen.

3. **Block Design:** Der Begriff kommt aus dem Telekommunikationsgebiet. Es modelliert eine Reihe von Modulen mit eigener Funktionalität, die miteinander durch gut definierten Schnittstellen verbunden sind. Diese Idee war auch im Software Design anwendbar. Durch die Kapselung des Programms und Daten könnte ermöglicht werden ,daß diese Fehler keine allzu großen Auswirkungen auf das System haben. Das Block Design soll im OOSE eine größere Veränderlichkeit ermöglichen.

2.2. Systementwicklung nach Jacobson et al.

Die Entwicklungsprozeß als Ganzes wird im OOSE in Form einer Pyramide, die in der Bauindustrie sich bewährt hat, erklärt:

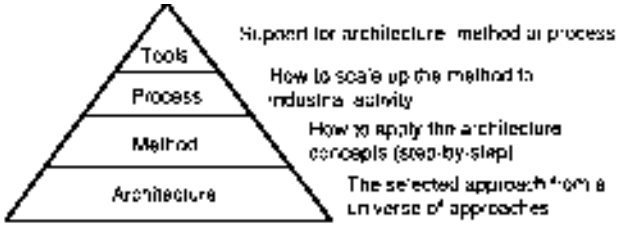

Abb. 1: Systementwicklung nach Jacobson et al. (Jacobson 1992, 29)

Demnach wird eine Methode aus einer Architektur entwickelt und aus der Methode heraus können der Prozeß und die geeigneten Tools entworfen werden. Unter Architektur im OOSE wird die Basis oder auch die Bedeutung aller entwickelten Modelle verstanden. Sie ist das Ergebnis der Anwendung einer Methode auf ein System. Es wird hier eine Parallele zur objektorientierten Entwicklung gezogen: die Architektur soll die beschreibende Klasse aller erzeugbaren Modelle oder Systeme (Instanzen) darstellen. Eine Methode beschreibt im OOSE , wie mit diesen Modellen in einer ´idealen´ Entwicklung zu arbeiten ist. Der Prozeß beschreibt das komplette Management eines Produkts in seinem ganzen Lebenszyklus. Die Entwicklung des Tools soll zum einen dazu dienen dem „industriellen Prozeß „ zu managen und zum anderen die Arbeit zu unterstützen.

2.3. Allgemeine Vorgehensweise

Der Entwicklungsprozeß im OOSE wird losgelöst vom Projekt eher produktbezogen betrachtet. Die Einzelprozesse werden im folgenden erklärt.

- **Analyseprozeß:** In dem Analyseprozeß soll im OOSE, das zu entwickelndes System (Software-System) konzeptuell dargestellt werden. Die Grundlage für diesen Prozeß sind hauptsächlich die Anforderungen an das System. Die Ziele dieses Prozesses sind, das System zu verstehen und eine gute Struktur für das System zu finden. Sie sollen gegenüber den Veränderungen robust gemacht und in klare, verständliche und unteilbare Teile geteilt werden. In diesem Prozeß werden zwei Modelle entwickelt: *Das Anforderungsmodell* und das *Analysemodell* (siehe 2.4.). Die Anforderungen werden zunächst in einem Anforderungsmodell und danach in einem Analysemodell überführt, ohne die spätere Implementierung zu berücksichtigen.

- **Konstruktionsprozeß:** In dem Konstruktionsprozeß wird das System aus den Modellen entwickelt, die in dem Analyseprozeß erzeugt worden sind (Anforderungs- und Analysemodell). In diesem Prozeß werden zwei Modelle erstellt: *Das Desigmodell* und *Implementationsmodell* (siehe 2.4.).

- **Komponentenentwicklungsprozeß:** In diesem Prozeß werden, die in dem Konstruktionsprozeß zu benutzenden Komponenten[3] entwickelt und gewartet. Im OOSE ist der Komponentenprozeß nicht zu einem spezifischen Produkt gebunden, sondern es soll einen „multi-produkt-prozeß“ darstellen.

.

- **Testprozeß:** In diesem Prozeß werden die Use-Cases und das ganze System getestet und bestätigt. Es wird versucht das Testen parallel mit den anderen Aktivitäten durchzuführen. Das Testen wird als ein Produkt, meist unabhängig von der eingesetzter Entwicklungsmethode betrachtet. Die grundlegenden Konzepte sind dabei die Testspezifikation und das Testergebnis. Es wird eine Verifikation und eine Validation durchgeführt. Die Verifikation überprüft, ob die

[3] Komponenten sind nach Jacobson et al. vorfabrizierte Einheiten, die man zusätzlich zu den Elementen der Programmiersprache oder als Standard – Bausteine bei der Entwicklung von verschiedenen Applikationen nutzen kann.

Ergebnisse mit der Spezifikation übereinstimmen und die Validation , ob die Ergebnisse den Benutzeranforderungen entsprechen. Validation wird im OOSE hauptsächlich durch die Anforderungsanalyse mit Einbeziehung der Benutzer und durch die Benutzung von Prototypen abgedeckt. Dabei soll sich das Use-Case-Konzept sehr gut für die Validationszwecke eignen. In diesem Prozeß werden folgende Aktivitäten durchgeführt:

- **Testplanung:** Die Testplanung kann sehr früh in dem Entwicklungsprozeß beginnen (meistens während der Analyse). Hier wird die Umgebung für das Testen definiert und Entscheidungen wie z.B. ob der Test automatisch oder manuell erfolgen soll, getroffen.
- **Testidentifikation:** Es wird hierbei identifiziert, was getestet werden soll.
- **Testspezifikation:** Innerhalb dieser Aktivität werden zunächst der Test und seine Zwecke beschrieben und später festgelegt, wie die Testfälle auszuführen sind.
- **Testausführung:** Ausführung der Testfälle und Analyse der Testergebnisse.
- **Fehleranalyse:** Analyse der Test und Identifizierung der Gründe für die gefundenen Fehler.
- **Testabschluß:** Diese Aktivität beinhaltet die Dokumentation der Testaktivitäten.

In der Abbildung 2 werden die drei Hauptprozesse Analyse-, Konstruktions- und Testprozeß dargestellt. Diese Prozesse sollen während der Entwicklung eines Produkts miteinander kommunizieren. Der Komponentenentwicklungsprozeß kommuniziert überwiegend mit dem Konstruktionsprozeß. Hierbei ist zu erwähnen, daß Jacobson et.al. die Erstellung und Nutzung der Komponenten außerhalb ihrer Vorgehensmodell beschreiben, die jedoch für den Gesamtprozeß eine wichtige Rolle spielen.

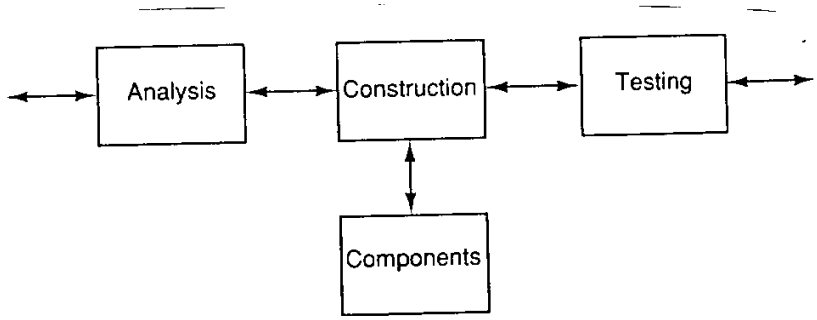

Abb. 2: Die Hauptprozesse und ihre Beziehungen im OOSE (Jacobson 1992, 120)

2.3.1. Vorgehensweise bei der Modellbildung im OOSE

Es werden im OOSE fünf verschiedene Modelle erstellt, von denen jedes bestimmte Aspekte des Systems berücksichtigen soll. Modelle, die im OOSE erstellt werden, sollen alle Phasen des Entwicklungsprozesses abdecken. Der Entwicklungsprozeß wird im OOSE als Vorgehen verstanden, bei dem Modelle ineinander überführt werden. Grundsätzlich gibt es dabei zwei Möglichkeiten:

operational: Ein Modell wird als Ganzes in ein anderes Modell überführt, welches dann auf Korrektheit geprüft wird.
transformational: Einzelne Transformationen werden auf einer Ebene der Einzelbestandteile eines Modells durchgeführt, wobei die Transformationen jeweils geprüft werden sollen.

Im OOSE wird hier vornehmlich operational gearbeitet. Dabei wird verstärkt auf Durchgängigkeit (treacebility) geachtet, so daß sich Objekte über mehrere Umwandlungen hinweg verfolgen lassen.

2.4. Modelle der OOSE

Wie schon oben erwähnt, werden im OOSE fünf verschiedene Modelle gebildet. Diese Modelle werden für ein Produkt nacheinander (wasserfallartig) erstellt, wobei auch Iterationen erlaubt sind. Diese Modelle sind:

2.4.1. Anforderungsmodell (Requierements modell)

Die erste Transformation erfolgt im OOSE von der Anforderungsspezifikation zur Anforderforungsmodell. Dieses Modell soll die Funktionalität des Systems definieren und das System abgrenzen. Dabei wird vom Standpunkt des Benutzers ausgegangen. Das Modell besteht aus folgenden Teilen:

- **Use-Case-Modell** (Beschreibung der Funktionalität des Systems)

Das Use-Case-Modell im OOSE benutzt zwei Konzepte: Akteure und Use-Cases. Ein Use Case beschreibt eine Interaktion eines Akteurs mit dem System. Es beinhaltet alle Einzelereignisse, die bei der Interaktion durchgeführt werden. Diese Einzelaktivitäten lösen jeweils einen Zustandswechsel des Gesamtsystems aus. Die Summe aller Use Cases beschreibt die Gesamtfunktionalität des Systems. Im OOSE wird zwischen Akteure und Benutzer unterschieden. Ein Benutzer ist die aktuelle Person, die das System benutzt, wogegen ein Akteur eine bestimmte Rolle repräsentiert, die ein Benutzer einnehmen kann. Ein Akteur wird als eine Klasse und die Benutzer als Instanzen von dieser Klasse betrachtet. Ein Beispiel in diesem Zusammenhang wäre ein Bankangestellter, der gleichzeitig auch Kunde seiner Bank ist. Dieser kann das System der Bank entweder in seiner Rolle als Bankangestellter oder in seiner Rolle als Kunde benutzen und verkörpert dabei entweder den Akteur Bankangestellter oder den Akteur Kunde.

Use Cases werden ausgehend von Akteuren identifiziert. Dabei wird für jeden Akteur festgelegt, welche Ereignisse er erzeugen kann. Die Ereignisse, die diesen initialen Ereignissen folgen, werden zu einem Use Case zusammengefaßt und beschrieben.
Folgende Fragen könnten dabei interessant sein:

•Welches sind die Hauptaufgaben eines Akteurs?

•Muß der Akteur auf Systeminformation zugreifen?
•Informiert der Akteur das System über die externe Änderungen?
•Wird der Akteur über unvorhergesehene Ereignisse informiert?

Beispiel für ein Use-Case-Modell

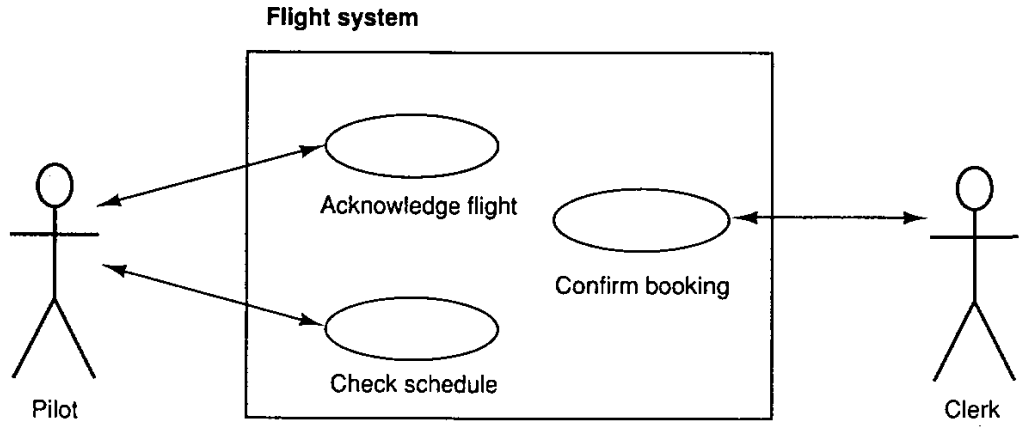

Abb. 3: Beispiel für ein Use-Case-Modell. Das System wird durch das Rechteck abgegrenzt. Jeder Akteur wird durch einen Person außerhalb des Rechtecks und die Use-Cases werden dagegen als Ellipsen innerhalb des Systems dargestellt. (Jacobson 1992, 128)

Das Anforderungsmodell wird zur Unterstützung der Wiederverwendung und zur Vorbereitung der Umwandlung in das Analysemodell weiter verfeinert. Dabei werden die gemeinsamen Teile von Use-Cases als eigenständige, abstrakte Use-Cases definiert. Die konkreten Use-Cases benutzen dann diese abstrakte Use-Cases. Außerdem werden die gemeinsamen Teile von Akteuren ebenso als abstrakte Akteuren beschrieben.

- **Interface-Beschreibungen** (Beschreibung der Interaktionen mit dem System)

Neben der Beschreibung der Funktionalität wird im OOSE die Interaktion mit dem System in Interface-Beschreibungen konkretisiert. Hierbei werden die Benutzerschnittstelle (graphisch, Kommondozeile usw.) beschrieben.

- **Modell des Problembereichs** (Basis für alle Beteiligten)

Im OOSE wird insbesondere bei ungenauen Anforderungsbeschreibungen zusätzlich versucht, ein logisches Modell des Problembereichs zu erstellen. Es soll ein gemeinsames Vokabular schaffen und beim Verstehen des Problembereichs helfen. Außerdem soll es bei der Identifikation von Use Cases durch Beschreibung der Dinge, die im System bearbeitet werden, helfen.

Im OOSE, wenn die Anforderungen nicht gut genug definiert sind, wird eine sog. Enterprise development (Unternehmensentwicklung) durchgeführt. Dabei wird die Organisation aus verschiedenen Perspektiven betrachtet. Das Ziel dabei soll die Problemgebiete identifizieren und alternative Lösungen vorzuschlagen sein. Die Unternehmensentwicklung wird im OOSE in folgender Phasen eingeteilt*:*

- **Gegenwärtige Zustand ermitteln (Establish current state):** Hier wird das gegenwärtige Unternehmen einschließlich mit seinen Zielen und Problemen untersucht. Am Anfang werden die verschiedenen Aktivitäten innerhalb der Unternehmen getrennt und analysiert, um herauszufinden, inwieweit die einzelnen Aktivitäten dem gesamten Unternehmen beitragen.
- **Unternehmensanalyse (Enterprise analysis):** Eine ideale Analyse Modell von dem gegenwärtigen Unternehmen wird erstellt.
- **Änderungsanalyse (Change analysis):** Es werden die gegenwärtigen Probleme und Bedürfnisse detailliert beschrieben und angemessene Veränderungen vorgeschlagen.
- **Unternehmensmodellierung (Enterprise design):** Die Ergebnisse der Phase 3 sollen in dieser Phase eine Beschreibung für die technische Realisierung bieten.
- **Verifikation und Validierung des Unternehmens (Enterprise verificatio and validation):** In dieser Phase wird die neu entwickeltes Unternehmen verifiziert und validiert.

Die Enterprise Design Ergebnisse sollen im OOSE als Eingabedaten für die Systementwicklung dienen. Es soll eine Anforderungsspezifikation für das zu entwickelndes System bieten.

2.4.2. Das Analysemodell

Das Anforderungsmodell wird im OOSE in ein Analysemodell überführt. Dieses Modell wird aus dem Use -Case-Modell abgeleitet und ist ein logisches Modell des Systems, daß auf stabilen Anforderungen und einer idealen Umgebung aufbaut
Die Strukturierung des System soll im OOSE durch die Modellierung von drei Objekttypen erfolgen:

- **Entity-Objekte:** In Entity-Objekten werden Informationen abgelegt, die im System über längere Zeit hinweg gehalten werden. Sie existieren über mehrere Abläufe von Use Cases hinweg und entsprechen weitgehend den Objekten, die im Modell des Problembereichs identifiziert werden konnten.
- **Schnittstellen-Objekte:** In diesem Objekttyp wird die Funktionalität behandelt, welche an den Systemgrenzen erbracht werden. Um diese zu identifizieren, gibt es im OOSE drei Möglichkeiten:

Direkte Übernahme der Objekte aus der Interface-Beschreibungen,
Betrachtung der Interaktion der Akteure mit dem System,
Betrachtung der interfacespezifischen Teile des Use Cases.

- **Kontroll-Objekte:** Kontroll-Objekte verbinden Interface- und Entity-Objekte und können direkt aus den Use Cases abgeleitet werden. Ein Kontroll-Objekt sollte an möglichst wenigen Use Cases beteiligt sein. Kontroll-Objekte sollen die Modifikation des Systemverhalten erleichtern.

Bei der Transformation des Anforderungsmodell zum Analysemodell wird das in den Use Cases beschriebene Systemverhalten auf die Objekte dieser drei Typen aufgeteilt. Jede dieser Objekttypen haben ihren eigenen Zweck und sollen einen bestimmten Aspekt des Systems modellieren. Es werden je nach Größe des Projekts eine Vielzahl von Analyse Objekten spezifiziert. Um die Anzahl der Objekte im Überblick zu halten, werden in der OOSE Subsysteme gebildet. Das System besteht demnach aus Subsystemen, die wiederum Subsysteme enthalten können. Die unterste Ebene des Subsystem sind veränderbare Einheiten, die „service packages“ genannt werden.

2.4.3. Das Designmodell

Das Designmodell dient zur Verfeinerung und Anpassung der Anforderungs-und Analysemodelle an die Systemumgebung (Datenbank, Programmiersprache, Netzwerk usw.) In diesem Modell wird das Analysemodell in ein implementierbares Design umgewandelt.
Das Ziel hierbei ist das Analyse Modell soweit zu verfeinern, daß die Sourcecode sehr leicht aus diesem Modell abgeleitet werden kann. Es wird jeden Objekt in dem Analysemodell einen Block in der Designmodell zugewiesen.
In diesem Modell wird das sog. Blockkonzept benutzt. Dieses Konzept beschreibt, wie der Code strukturiert, bewältigt und geschrieben werden soll. Die Blöcke sind eine Abstraktion der aktuellen Implementation des Systems und kommunizieren über Botschaften (stimuli). Die Darstellung dieser Kommunikation erfolgt durch die Interaktionsdiagramme. Es wird für jedes Use-Case ein Interaktionsdiagramm gezeichnet.
Beispiel für ein Interaktionsdiagramm:

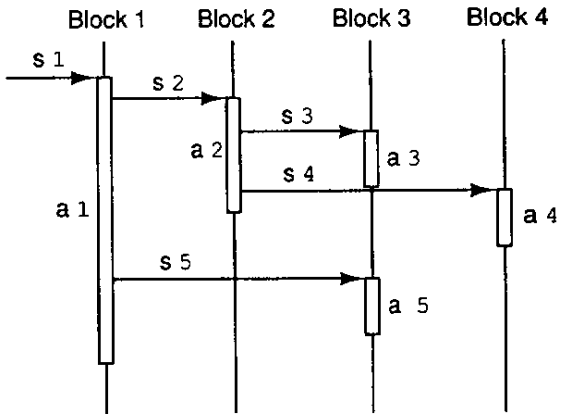

Abb. 4: Beispiel für ein Interaktionsdiagramm. Das Diagramm zeigt wie Botschaften (s) zwischen den Blöcken gesendet und wie dadurch die Aktivitäten (a) in diesen Blöcken ausgelöst werden. (Jacobson 1992, 148)

2.4.4. Das Implementierungsmodell

Das Designmodell bildet die Basis für dieses Modell. Das Implementierungsmodell ist im OOSE der eigentliche Programmcode. In diesem Modell werden die in dem Designmodell spezifizierten Objekte Blöcke) implementiert. Diese Blöcke können bei nicht komplexen Umgebungen, den Konzepten der gewählten Programmiersprache (z.B. Klasse, Module usw.) entsprechen. Die Auswahl der Programmiersprache für Implementation wird im OOSE offen gelassen. Es wird eine objektorientierte Sprache empfohlen, es kann jedoch auch eine andere Sprache für die Implementation gewählt werden.

2.4.5. Testmodell

Das Testmodell ist das letzte Modell, das im OOSE entwickelt wird. Es beschreibt im allgemeinen die Ergebnisse des Testprozesses (vgl. hierzu Testprozeß im Abschnitt 2.3.)

2.5. Managementaspekte vs. Methodische Aspekte im OOSE

Jacobson et.al. versuchen die methodischen Aspekte von den Management-Aspekten zu trennen und führen aus diesem Grunde nebeneinander ein Prozeßmodell und ein Projektmodell ein. . Die beiden Modelle sind linear und laufen in der Regel genau einmal ab.

Das Prozeßmodell (System Life Cycle) soll die methodisch-technische Seite abdecken und es besteht aus den Modellen, die in den Hauptprozessen erstellt werden (vgl. Abschnitt 2.2.3.).

Auf der anderen Seite konzentriert sich das Projektmodell auf eine Phaseneinteilung und die Definition von Meilensteinen. Die Phasen in dem Projektmodell sind folgendermaßen benannt :

1. Vorstudie (Pre-study),
2. Machbarkeitsstudie (Feasibility study),
3. Etablierung (Establishment),
4. Ausführung (Execution),
5. Abschluß (Conclusion).

Das Prozeßmodell läßt sich dann nach Jacobson et.al. in das Projektmodell einbetten, in dem man die Ergebnisse der Modelle, die in den Prozessen erzeugt werden mit Projektmodell-Meilensteinen identifiziert.

Nachdem in diesem Kapitel die Methode OOSE von Jacobson et.al dargestellt wurde, soll im nächsten Kapitel das NIMSAD-Framework von Nimal Jayaratna erläutert werden, mit dem Ziel die OOSE-Methode zu evaluieren.

3. NIMSAD-Framework

NIMSAD steht, wie schon erwähnt, für **N**ormative **I**nformation **M**odel-based **S**ystems **A**nalysis und **D**esign. Es wurde von Nimal Jayaratna aus der Notwendigkeit heraus entwickelt, ein methodenunabhängiges Framework zu haben, um die Ziele von Methoden untersuchen zu können. Es wurde aus den Erfahrungen, die aus der industriellen Praxis, der Beratungen und der theoretischen Forschung basieren, heraus entwickelt. Was dieses Framework leisten soll, drückt Jayaratna im Vorwort seines Buches folgendermaßen aus: The role of this book is [17, S. xii] :

- to help ist readers and those using methodologies to break free from political constraints
- to raise their consciousness to consider the desirabilitiy of their actions before their feasibility
- to consider the effect of their actions on others as well as on themselves
- to enable more open and facilitating debate and discussion
- to minimize the desire to achive self-needs sat the expense of others. Instead, such achievements can be accomplished in the context of others without destroying their hopes and opportunities.

Diese Worte von Jayaratna betonen seinen Ansatz, daß das NIMSAD-Framework als ein Instrument zur Diskussion, zum Lernen und zum kritischen Denken dienen soll. Er weist in seinem Buch darauf hin, daß das NIMSAD – Framework kein „Kochbuchrezept“ darstellen soll , wie eine passende Methode gefunden werden kann, sondern es soll bei der Lösung von Problemen ein generelles Verständnis zu entwickeln, Methoden in ihrer Struktur, ihren Schritten und ihrer Form zu evaluieren und daraus Schlüsse zu ziehen, helfen.
NIMSAD ist ein Framework mit dem es möglich sein soll, alle Methoden (nicht nur Informationssystem-Methoden), die zur Lösung von Problemen herangezogen werden, zu evaluieren und auszuwerten. Einer von den Zielen dieser Studienarbeit ist auch zu zeigen, ob dies auch für die Methode OOSE gilt.

3.1. Framework vs. Methodologie

Bevor das NIMSAD – Framework näher erläutert wird, sollen zunächst die Begriffe „Methodologie“ und „Framework“ aus der Sicht des Autors beschrieben werden. An dieser Stelle ist zu erwähnen, daß der Autor sich dem Sprachgebrauch im Bereich der Informationssysteme anpaßt und somit nicht zwischen den Begriffen „Methode“ und „Methodologie“ unterscheidet. Dies könnte zur Verwirrungen führen, da diese beiden Begriffe im Deutschen eine unterschiedliche Bedeutung haben. Im weiteren dieser Studienarbeit wird der Begriff „Methode“ verwendet.

Jayaratna definiert eine Methode bzw. Methodologie als:

„ an explicit way of strukturing one's thinking and actions. Methodologies contain models and reflect particular perspektives of `reality´ based on a set of philosophical paradigmes. A Methodology must show ‚what' steps to take, ‚how' those steps are to be performed (...) the reasons ‚why' the methodology user must follow those steps and in the suggested order." [17, S.35ff.]

Und ein Framework als:
„ (...) meta-level model through which a range of concepts, models, techniques, methodologies can either be clarified, compared, categorized, evaluated and/or integrated." [ebd. S.43]

Nach Jayaratna kann ein Framework mit einer Linse verglichen werden, durch die ein Beobachter eine Reihe von Konzepten, Modellen, Techniken und Methoden betrachten, verstehen und beurteilen kann. Die Methodologie (nach Jayaratna), bzw. eine Methode soll hingegen aufzeigen, welche Schritte wie und warum in einer bestimmten Reihenfolge vollzogen werden sollen. Der wesentliche Unterschied zwischen einer Methode und einem Framework ist nach ihm, daß eine Methode immer eine zeitabhängige Ordnung von Denk- und/oder Aktionsschritten impliziert.

3.2. Informationssystementwicklung vs. Softwareentwicklung

An dieser Stelle werden die Begriffe „Informationsystementwicklung" und „Softwareentwicklung " aus der Sicht von Jayaratna beschrieben, da diese beiden Begriffe in der Informatik meistens austauschbar benutzt werden [17]. Der Begriff System wird in der Wissenschaft Informatik sehr oft verwendet; es gibt jedoch keine genauen Definitionen[4].

Die Softwareentwicklung oder Software-Engineering beschäftigt sich nach Jayaratna nur mit der Konstruktion von Software und mit der Qualitätssicherung der Software. Für computer-unterstützte Informationssysteme sind jedoch auch organisatorische Aspekte von großer Bedeutung, die nach seiner Ansicht nicht genügend von der Softwareentwicklung abgebildet werden.
Den Unterschied zwischen diesen beiden Begriffe drückt Jayaratna folgendermaßen aus:

> „The task of software engineering is to produce software to match the quality and needs of a particular *software specification*, while that of systems devolopment is to match the quality ans information needs of a particular *user*(*s*)." [ebd. S.34]

3.3. Anwendung des NIMSAD-Frameworks bei der Evaluation

Das NIMSAD Framework basiert auf vier Elementen, die nach Jayaratna in jedem Problemlösekontext existieren und aus diesem Grunde auch bei der Bewertung von Methoden

[4] An dieser Stelle wird auf [Klischewski97, Kapitel 3] verwiesen. Dort befindet sich eine ausführliche Beschreibung und Auseinandersetzung mit dem Systembegriff.

berücksichtigt werden sollten: Problemlöse-Situation (Methodenkontext), gewählter Problemlöser (Methodenbenutzer), Problemlöseprozeß (Methode) Evaluation der oben genannten drei Elemente.

Nach Jayaratna sollen die Elemente und die Schritte von NIMSAD nicht als ein Idealtyp übernommem werden, sondern sie sollen dazu dienen, sowohl die Methoden als auch die eigene Handlung kritisch zu überdenken. Aus diesem Grund wird im Rahmen des NIMSAD-Frameworks als Mittel zur Evaluation ein Fragenkatalog entwickelt, mit dessen Hilfe der Methodenanwender/innen die Methoden und die Beziehungen der Schritte innerhalb der Methode überdenken können.
In der Tabelle 1 wird ein Ausschnitt aus dem Fragenkatalog dargestellt.

PROBLEMSITUATION	METHODENANWENDER	PROBLEMLÖSESITUATION
Wie hilft die Methode ihren Benutzer die Klienten usw. in den Problemlöseprozeß zu bringen? Akzeptiert die Methode Auftraggebers Bedürfnisse als den Anfangspunkt? Welche Situationen sind geeignet für den Einsatz der Methode? Was ist der Zweck der Methode? Wozu ist sie fähig? Welche philosophische Sicht hat die Methode?	Was erfordert die Methode von ihren Anwendern vor der Anwendung? Welche Modelle bietet die Methode für ihre Anwender um die Situation zu verwalten? Welche Rolle erwartet die Methode von Ihren Anwendern? Welche Bedingungen gibt es für eine erfolgreiche Benutzung der Methode?	**Fragen zum Schritt 1:** Wie hilft die Methode bei der Abgrenzung der Situation? Welche Rolle hat der Klient? Betont die Methode die Aufnahme und/oder die Partizipation der Klienten? Bietet die Methode Kriterien für die Auswahl der Informationen? **Fragen zum Schritt 2:** Welche Modelle und Techniken bietet die Methode um die Situationseigenschaften auszudrücken? Befürwortet sie konzeptuell/logische oder technische Ausdrucksweise? Macht sie einen Unterschied zwischen beiden? **Fragen zum Schritt 3:** Bietet die Methode eine Hilfe bei dieser Definition? Wenn nicht, welche Prognose akzeptiert sie als legitim? Oder gibt es mehrere Prognosen? **Fragen zum Schritt 4:** Mit welchen Problemen oder Problemtypen beschäftigt sich die Methode? Welche Kriterien bietet sie für die Problemdefinition? **Fragen zum Schritt 5:** Leitet die Methode das gedankliche System aus der Problemdefinition her oder nimmt sie es als gegeben an? Wie werden die Veränderungen gehandhabt? **Fragen zum Schritt 6:** Welche Fähigkeiten für das Design erwartet die Methode von ihren Anwendern? Welche logische Konzepte bietet sie für die Strukturierung der Lösungen? Welche Techniken bietet sie für das Design? **Fragen zum Schritt 7:** Welche Modelle oder Techniken bietet die Methode? Welches Wissen und Fähigkeiten erwartet sie von ihren Anwendern? **Fragen zu Schritt 8:** Wie hilft die Methode bei der Implementation des Designs? Macht die Methode ihren Anwendern auf die verschiedenen Implementationsstrategien aufmerksam?

Tabelle 1: Fragenkatalog für die Evaluation der Methoden nach NIMSAD

3.4. Elemente des NIMSAD-Framework

In der Abbildung werden die Elemente und die Schritte des NIMSAD-Frameworks dargestellt. Die einzelnen Elemente werden dann im folgenden genauer erklärt, da sie für die Bewertung der OOSE-Methode im Kapitel 4 von Bedeutung sind.

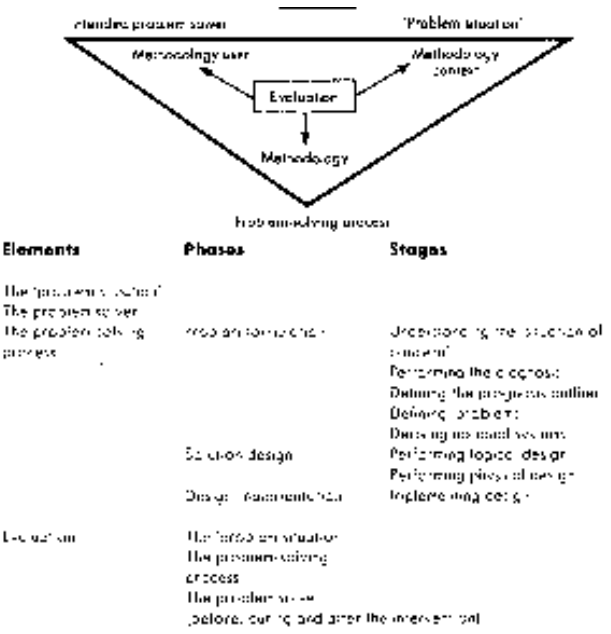

Abb. 5: Die Elemente und Schritte des NIMSAD-Frameworks (Jayaratna 94, S.128)

3.4.1. Element 1: Problemlöse-Situation (Methodenkontext)

Die Problemsituation ist der Kontext, indem eine Methode eingesetzt wird. Dieser Kontext ist nach Jayaratna meistens eine Organisation. Der Grund für die Betrachtung dieses Elements in NIMSAD ist, daß der gesamte Methodenkontext (die Benutzer, andere Organisationsmitglieder, die zwischenmenschliche Beziehungen, räumliche Gegebenheiten, Teilsysteme und ihr Zusammenspiel mit anderen Teilsystemen) verstanden werden muß, um benutzbare Softwaresysteme zu entwickeln und die Effektivität des Softwaresystems beurteilen zu können. Je größer das Wissen des Problemlösers über die Organisation ist, desto größer ist sein Verständnis für die wirklichen Probleme der Organisation.

3.4.2. Element 2: Problemlöser (Methodenanwender)

Ein Problemlöser (Methodenanernder) kann z.B. ein Unternehmensberater, Systemanalytiker oder ein Systemdesigner sein. Der Problemlöser hat nach Jayaratna einen entscheidenden Einfluß auf die Umsetzung und den Erfolg einer Methode, da die Effizienz und die Effektivität einer Methode von dem Problemlöser abhängt. Für den Erfolg und für den richtigen Einsatz einer Methode ist entscheidend

die wichtigen Elemente der Problemsituation auszuwählen. Jayaratna weist darauf hin, daß einige dieser Elemente mit Hilfe der Methoden und andere dagegen mit dem sog. mentalen Konstrukt des Methodenanwenders selektiert werden. Das Konzept des mentalen Konstrukts wird im Rahmen des NIMSAD-Frameworks behandelt. Dadurch möchte Jayaratna die Methodenanwender/innen auf die Wichtigkeit des kritischen Denkens aufmerksam machen und sie zur bewußten Handlungen bei der Auswahl der Methoden hinführen.

Bestandteile des mentalen Konstrukts sind: Wahrnehmungsprozeß, Wertesystem, Ethik, Motive, Vorurteile, Erfahrungen, Abstraktionsfähigkeit, Wissen und Fähigkeiten, Strukturierungsprozeß, Rolle und Modelle und Frameworks. Die einzelnen Bestandteile stehen in einer dynamischen Interaktion und werden durch ständige Reflexion und äußere Eindrücke beeinflußt.

In diesem Zusammenhang weist Jayaratna darauf hin, daß das durch mentales Konstrukt beeinflußte Wahrnehmung von **intellektuellen** und **politischen** Denkprozessen dominiert wird.

Nach ihm veranlassen die politischen Denkprozesse, die Herbeiführung persönlich vorteilhafter Ergebnisse (wie z.B. die Anhäufung von Macht und Kontrolle über Andere) aus Prozessen. Diese werden durch Manipulation (Überreden, Macht ausüben) dazu gebracht, den eigenen Zielen zu dienen. Im Zusammenhang mit Methoden weist er darauf hin, daß diese Prozesse die Kreativität und Entfaltung der Einzelnen unterdrücken und damit einen schädlichen Einfluß auf den Erfolg einer Methode haben könnten.
Intellektuelle Denkprozesse dagegen haben das Ziel, andere durch Überzeugung zum Mitmachen zu bewegen. Hierdurch können nach Jayaratna aufgrund der Überzeugungsarbeit neue Gedankengänge entstehen, die zur Abschätzung der Folgen für eine Handlung führen könnten.

3.4.3. Element 3: Problemlöseprozeß (Methode)

Die Schritte dieses Elementes sollen nicht als Idealtyp verstanden werden, um die Methoden zu bewerten, sondern sollen vielmehr zu ihrem Verständnis über die Methoden (z.B. zu fragen, ob die Methoden diese Schritte ausführlich betrachten oder ignorieren; warum sie dies tun und wie sie diese Schritte kombinieren bzw. voneinander trennen usw.) beitragen.

3.4.3.1. Schritt 1: Die maßgebliche Situation verstehen

Der maßgebliche Situation ist der Teil, der für die Lösung des Problems von Interesse ist. In diesem Schritt soll der Problembereich abgegrenzt werden. Diese Grenze soll bestimmen, welche Informationen in welchem Umfang gesammelt werden

3.4.3.2. Schritt 2: Eine Diagnose durchführen

Die Diagnose soll helfen, das Verständnis über die maßgeblichen Situation zum Ausdruck zu bringen, d.h. es wird in diesem Schritt eine sog. Ist-Analyse durchgeführt (wo stehen wir jetzt?). Die Durchführung der Diagnose soll Verständnislücken oder Mißverständnisse erkennen und diese mit

den Auftraggebern klären und beseitigen, ermöglichen. In diesem Zusammenhang sollen die Methodenanwender die Relevanz und das Wesen der Techniken und Tools (wie z.B. Datenflußdiagramme usw.), die von den Methoden angeboten werden, untersuchen.

Im Rahmen der NIMSAD werden zur Formulierung der Diagnose zwischen

- conceptual/logical expression (*Diagnose Modell 1, DM1*) und
- physical expression (*Diagnose Modell 2, DM2*)

unterschieden.

Im *Diagnose Modell 1(DM1)* wird die „was/welche" Fragestellung beschrieben (wie z.B. welche Rollen, Aktivitäten, Entscheidungen kommen in der maßgeblichen Situation vor?)
Im *Diagnose Modell 2 (DM2)* wird die „wie" Fragestellung beschrieben (z.B. wie werden die Funktionen, Aktivitäten, Entscheidungen unterstützt bzw. durchgeführt.

3.4.3.3. Schritt 3: Konturen der Prognose herausstellen

In diesem Schritt wird der gewünschte Zustand für die gegenwärtige maßgebliche Situation definiert, d.h. hier wird eine sog. Soll-Analyse durchgeführt (wo wollen wir hin und warum?).
Es werden hierbei nur die Umrisse und Gründe für die Prognose definiert. Die Beschreibung des Inhalts der Prognose ist wiederum die Aufgabe der Design Phase. In diesem Schritt soll die kritische Untersuchung der Gründe für den gewünschten Zustand im Vordergrund stehen.

3.4.3.4. Schritt 4: Hindernisse erkennen

Hier soll versucht werden zu verstehen, was den maßgeblichen Zustand hindert, sich zu einem gewünschten Zustand zu verändern. Nach Jayaratna liegt das Problem, das erkannt und formuliert werden muß zwischen Diagnose und Prognose. Aus diesem Grund sollen in diesem Schritt die Abwesenheit und/oder Anordnung der Elemente im Diagnose-Modell identifiziert und kritisch untersucht werden.

3.4.3.5. Schritt 5: Das gedankliche System herleiten

Das gedankliche System ist nach Jayaratna dasjenige System, daß entwickelt werden muß, wenn die auftraggebende Organisation die Probleme bzw. Hindernisse bewältigen kann. Es soll die maßgebliche Situation in die gewünschte Situation zu transformieren helfen. Das gedankliche System soll Richtlinien vorgeben, wie die erkannten Probleme (Hindernisse) beseitigt werden können. Die Beschreibung der Merkmale und das zu erwartende Verhalten des gedanklichen Systems wird von Jayaratna Sytemspezifikation oder Anforderungsspezifikation genannt. Das Ziel ist hierbei, daß die Methodenanwender/innen die Anforderungen der Auftraggeber tiefergehend analysieren um damit die Gültigkeit der Beschreibung der Prognose sowie dem gedanklichen System deutlich zu machen.

3.4.3.6. Schritt 6: Ein konzeptuelles/logisches Design erstellen

Die Aufgabe des konzeptuellen/logischen Design ist es, die Elemente (z.B. die Funktionen und Informationen) des gedanklichen Systems zu erzeugen oder anders zu gestalten. Die Innerhalb dieser

Schritt zu entwickelnden logischen Modelle sollen einfach und umfassend die logischen Elemente der maßgeblichen Situation darstellen. Logisch bedeutet dies in diesem Zusammenhang, daß die Elemente als wichtig und nützlich argumentiert werden können. Das Ergebnis in diesem Schritt soll die Erzeugung von einer vereinbarten und akzeptierten logischen Designspezifikation sein, die die Arten und Funktionen der Elemente der maßgeblichen Situation festlegen soll. Hierbei erwartet Jayaratna von den Methoden, daß sie die entsprechende Techniken für die Entwicklung des konzeptuellen/logischen Designs bereitstellen.

3.4.3.7. Schritt 7:Ein technisches Design erstellen

Das technische Design kann als die Überlegung und der Auswahl von Wegen und Mitteln zur Realisierung des logischen Designs betrachtet werden. Dadurch können verschiedene technische Design Modelle zur Realisierung der Merkmale für das gleiche logische Modell innerhalb der gegebenen technischen Ressourcen und Einschränkungen erzeugt werden. Diese könnten von den Auftraggebern formuliert werden, wie z.B. welche Produkte oder Tools zu benutzen sind usw.

Um die Design Elemente zu entwickeln werden Antworten auf die „wie“ Fragestellung gesucht (z.B. Wie ist das logische Modell in eine technische Form innerhalb gegebener Ressourcen zu übertragen?). Es wird auch Prognose Modell 2 (PM2) genannt. Das Prognose Modell 2 (PM 2) sollte die benutzte Technologie, Ein- und Ausgangsformate, Standards von Dokumenten, Datenspeicher Organisation und Zugriffsmethoden, beteiligte Personen usw. beinhalten.

3.4.3.8. Schritt 8: Implementation

Die Implementation beschäftigt sich mit der Realisation des gedanklichen Systems innerhalb des Kontext der maßgeblichen Situation. Es soll eine Art Test für die Beurteilung der Gültigkeit der Annahmen darstellen. Das Ausmaß der Implementation kann nach Jayaratna durch die Abbildung der zwei Modelle ermittelt werden, vorausgesetzt das Diagnose Modell gibt die maßgebliche Situation wirklichkeitsnah wieder. Der konzeptuelle Vergleich zwischen Diagnose Model1 (DM1) und Prognose Model 1 (PM1)soll die Unterschiede und damit auch das Ausmaß der Änderungen zeigen. Der Vergleich zwischen Diagnose Model 2 (DM2) und Prognose Model 2 (PM2) soll hingegen die Art der Elemente zu identifizieren helfen, die verändert werden müssen.

3.4.4. Element 4: Evaluation

Die Evaluation ist das letzte Element des NIMSAD-Frameworks. Sie hilft die Effektivität des Problemlöseprozesses und des Problemlösers in der Problemsituation zu messen. Nach Jayaratna kann keiner Problemlöseprozeß abgeschlossen betrachtet werden, bevor die Evaluation durchgeführt wurde. Seiner Meinung nach gibt es trotzdem nur sehr wenige Methoden, die die Evaluation als Methodenschritt beinhalten. Es werden die Elemente des NIMSAD-Framemeworks, (die Problemsituation, der Problemlöseprozeß und der Problemlöser) als Basis für die Evaluation ausgewählt. Hierbei wird nicht nur die Wichtigkeit dieser Elemente, sondern auch z.B. die

Beziehungen dieser Elemente untereinander betrachtet. Die Evaluation muß auf drei Stufen durchgeführt werden:

1. Bevor der Eingriff, um die Effektivität und Bemühungen zu erhöhen,
2. während der Systementwicklung, wegen der dynamischen Art der Elemente und
3. nach der Systemeinführung, so daß daraus Rückschlüsse gezogen werden.

3.5. Zusammenfassung des NIMSAD-Frameworks

In der Abbildung 5 wird das NIMSAD-Framework zusammenfassend dargestellt. Es werden in der Abbildung die drei Elemente dargestellt. Das vierte Element basiert auf diesen drei Elementen und wird deswegen nicht expliziert auf der Abbildung aufgeführt.

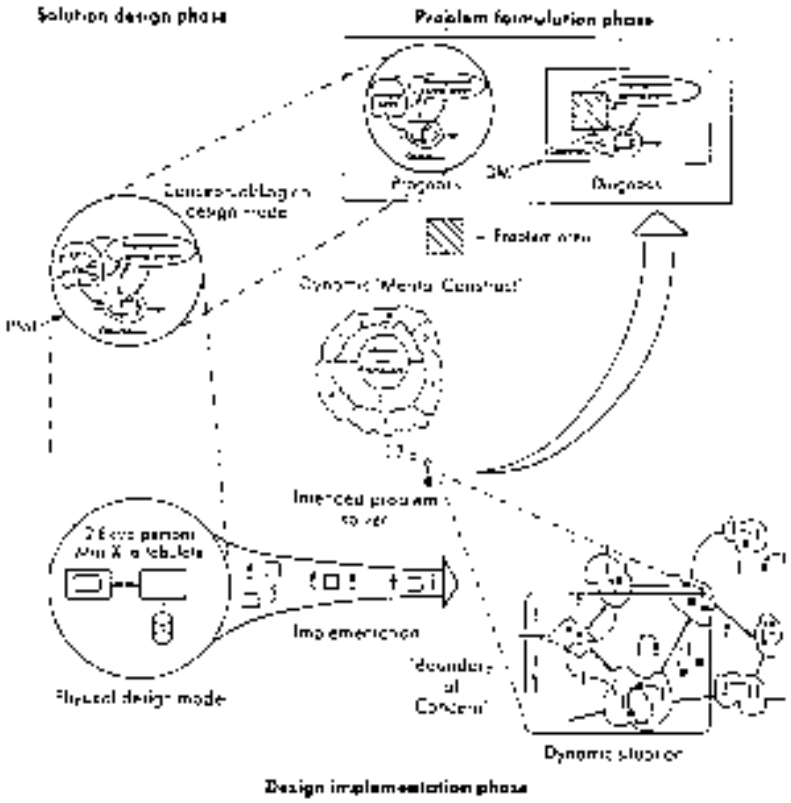

Abb. 6: Das komplette NIMSAD-Framework (Jayaratna 94, S.106)

Im nächsten Kapitel wird die Evaluation der Methode OOSE nach NIMSAD-Framework durchgeführt.

4. Kritische Beurteilung der OOSE nach NIMSAD

In diesem Kapitel dieser Studienarbeit wird die Methode OOSE (Object-Oriented Software Engineering) von Jacobson et al. mit Hilfe des NIMSAD (Normative Information Model-based Systems Analysis and Design) Frameworks von Nimal Jayaratna evaluiert. NIMSAD ist, wie schon im Kapitel 3 erwähnt, ein Framework, welches die kritische Untersuchung der Methoden im Vordergrund stellen will. Jayaratna will mit diesem Famework betonen, daß die Methodenanwendung nicht nur ein operatives Tun sondern vielmehr ein kontinuierlicher Lernprozeß darstellen soll.

Die Evaluation in dieser Studienarbeit soll einerseits zeigen, ob das NIMSAD-Framework für die Bewertung der Methode OOSE herangezogen werden kann und andererseits soll sie zu der Bewertung der OOSE-Methode führen.

Das NIMSAD-Framework, das im Kapitel 3 dargestellt wurde, besteht aus vier Elementen (die Problemsituation, der Problemlöser und Problemlöseprozeß und Evaluation). Aus diesem Grund wird die OOSE-Methode im Hinblick auf diese Elemente untersucht, ob und inwieweit die OOSE diese Elemente berücksichtigt. Dabei wird der im Rahmen des NIMSAD-Frameworks aufgestellte Fragenkatalog[5] auf die Methode angewendet. Hierbei ist zu erwähnen, daß die im Rahmen enthaltenen Fragestellungen im NIMSAD nicht beantwortet werden. Er soll vielmehr dazu dienen, daß die Methodenanwender/innen je nach Einsatz und untersuchende Methode sich diese Fragen aufs Neue stellen, Die um die Methode und ihre Handlung besser zu verstehen, bewerten und einsetzen zu können. Jayaratna weist aus diesem Grund ausdrücklich darauf hin, daß die in seinem Buch durchgeführte Evaluationen[6] nur von seinem Standpunkt heraus durchgeführt worden seien. Weiterhin erwartet er, daß die Methodenanwender/innen eigene Evaluation durchführen und eigene Schlußfolgerungen ziehen sollen.

Nachdem die Vorgehensweise der Evaluation in dieser Studienarbeit geschildert wurde, wird im weiteren mit der eigentlichen Bewertung der OOSE-Methode fortgefahren.

4.1. Die Problemsituation (Methodenkontext)

Wie schon im Kapitel 3.4.1. erläutert, ist die Problemsituation der Kontext, indem die Methode eingesetzt werden soll. Hierbei sollten nach Jayaratna die Methoden Hilfestellungen beim Verstehen der Gründe für die Bedürfnisse und Erwartungen der Auftraggeber in der Problemsituation bieten. Dies ist einerseits wichtig, um zum einen das zu lösende Problem richtig zu verstehen und es damit erfolgreich zu lösen, zum anderen die Effizienz des Methodeneinsatzes zu bewerten. Andererseits

[5] Siehe Abb. Im Kapitel 2 in dieser Arbeit.
[6] Nimal Jayaratna evaluiert in seinem Buch „Understanding and Evaluating Methodologies“ drei Methoden: Strukturierte Methode (De Marco, 1979), ETHICS Methode (Mumford 1983) und Soft Systems Methode (Checkland, 1981).

sollten Methodenanwendenden die ethischen und moralischen Verpflichtungen gegenüber den Beteiligten in der Organisation gerecht werden.

Jayaratna kritisiert in diesem Zusammenhang, daß die meisten Informationssystemmethoden sich nur mit der Herausfindung der Informationsanforderungen und nicht mit den wirklichen Geschehnissen in der Organisation als Ganzes, beschäftigen. Dies ist nach Jayaratna nicht genug um die Problemsituation zu verstehen, da die Gründe hierfür nicht hinterfragt und ohne Untersuchung der Gültigkeit dieser Gründe von den Auftraggebern übernommen werden.

> „At this Stage, (...) we are concerned with the need for an intended problem solver, to acquire as much knowledge as possible about the organisation as a whole and not just about those aspects that may be the concern of methodologies. For every aspect of the situation, that the methodologies does not offer help to comprehend or transform, the intended problem solver has to devise ways of coping with it.“ [17, S.59]

Dieses Zitat betont den Ansatz von Jayaratna, daß die Methodenanwender/innen sich mit der evaluiernden Methode auseinandersetzen und die Schritte in der Methode kritisch untersuchen müssen, um die Methode in der jeweiligen Situation einzuschätzen und geeignete Anpassung der Methode vornehmen zu können.

Jayaratna erwartet in diesem Zusammenhang von den Methodenentwicklern, daß

> „the methodolohy creators should explicitly consider what enviromental conditions/n setting are suitable for the application of their methodologies and what dimensions of the organisation they aim to transform. By doing so, the creators can help their methodology users to question what type of situations they face and what, if any, changes they should undertake to adopt the mehodology for that situation.“ [17, S.231]

Nachdem die Sichtweise von Jayaratna über die Problemsituation dargestellt wurde, wird im folgenden untersucht, ob und inwieweit die Methode OOSE den Methodenanwendenden hilft den Methodenkontext zu verstehen.

Jacobson et al. definieren ihre Methode-OOSE als

> „a method, which provides support for the creative design of software products, but which also provides an approach for making software development a rational industrial process. Thus the aim is not just to produce well-engineered software systems, but to make them viable ´living´ products that can be exploited in an industrial environment.“ [15, S.1]

Es kann aus diesen Sätzen gefolgert werden, daß die Entwickler der Methode-OOSE die Herstellung von Software getrennt von ihrem Einsatz sehen, da sie die Software als ein Produkt und den

Entwicklungsprozeß als einen Industriellen- bzw. Produktionsprozeß betrachten. Diese Sichtweise entspricht mehr oder weniger dem Produktionssicht[7] nach Floyd bei der Softwareentwicklung. Es kann demnach auch nicht von dieser Methode erwartet werden, daß sie eine gute Unterstützung beim Verstehen der Problemsituation bietet.

Es wird wie schon im Kapitel 2 vorgestellt im Rahmen der OOSE eine sog. Enterprise-development (Unternehmensentwicklung) durchgeführt, mit dem Ziel Problemgebiete zu identifizieren und alternative Lösungen vorzuschlagen. Diese Art der Analyse der Methodenkontextes könnte den Methodenanwendenden beim Verstehen der Problemsituation unterstützen. Da aber die Unternehmensentwicklung im OOSE nur dann durchgeführt wird, wenn die Anforderungen „nicht gut genug bekannt" sind, kann dies als eine Schwäche der Methode betrachtet werden. Hieraus kann gefolgert werden, daß wenn die Anforderungen seitens der Auftraggeber „gut genug" definiert sind, könnten diese ohne kritische Hinterfragung übernommen werden. In diesem Zusammenhang machen die Entwickler der Methode auch keine Aussage darüber, wie die Methodenanwender entscheiden sollen, wann die Anforderungen „gut genug" oder „schlecht" definiert sind. Dies würde bedeuten, daß jeder Methodenanwender/innen aufgrund seines spezifischen mentalen Konstruktes über die Notwendigkeit der Durchführung einer Unternehmensentwicklung unterschiedlich entscheiden könnte.

4.2. Der Problemlöser (Methodenanwender)

Die Wichtigkeit der Subjektivität der Methodenanwender/innen drückt Jayaratna in seinem Buch folgendermaßen:

> „Succes in practice depends not only on the methodology assistance, but rather on how its users structure their meaning of the situations and manage their interpersonal relationships with others." [17, S.234]

Im folgenden wird untersucht, ob die OOSE auch diese Sichtweise vertritt und ob sie ihren Anwender/innen auf das mentale Konstrukt aufmerksam macht.

.

Die Methode OOSE macht in keinen seiner Phasen bzw. Schritten den Methodenanwender/innen auf das mentale Konstrukt aufmerksam. Da wie im vorigen Abschnitt dargestellt wurde, vertreten die Entwickler dieser Methode die Produktionssicht bei der Herstellung von Software. Kennzeichnend für diese Sichtweise ist, daß die Herstellung der Software menschenunabhängig angesehen wird. Aus diesem Grund kann von OOSE auch nicht erwartet werden, daß sie den Methodenanwender/innen als einen entscheidenden Faktor in dem Softwareentwicklungsprozeß betrachtet und somit auch nicht auf das mentales Konstrukt ihrer Anwendern aufmerksam macht.. Diese Annahmen werden auch durch das folgende Zitat von Jacobson et al. bekräftigt :

[7] Vergleiche hierzu Abschnitt 1.2. in dieser Studienarbeit.

> „ Each person trained for an operation must perform it in similar manner. The process must yield a foreseeable result, irrespective which indivudual performed the job.“ [15, S.8]

Dieses Zitat zeigt, daß es nicht relevant ist, wer die OOSE-Methode anwendet. Es kann durch den Einsatz der Methode OOSE auf jeden Fall das selbe Ergebnis erzielt werden. Diese Annahme von den Entwicklern kann nicht übernommen werden, weil die Methodenanwender/innen die Schritte und die Modelle der Methode unterschiedlich interpretieren und benutzen könnten. Aus diesem Grund können die Entwickler der Methode nicht erwarten (auch wenn es eine Idealvorstellung ist), daß durch die Anwendung der OOSE immer die gleichen Ergebnisse erzielt werden können.

Wie im Kapitel 2 dargestellt, wird Systementwicklung im OOSE als Modellbildung verstanden. Es wird aber bei der Bildung dieser Modelle nur darauf hingewiesen, daß die Entwicklung dieser Modelle talentierte Entwickler erfordert, aber nicht genau erklärt, was zum einen in diesem Zusammenhang talentiert bedeutet und zum anderen, welche Talente die Methodenanwender dafür haben sollen.

4.3. Problemsituation (Methode)

4.3.1. Die maßgebliche Situation verstehen

In diesem Abschnitt wird untersucht, wie die Methode OOSE bei der Abgrenzung und Verstehen der maßgeblichen Situation hilft.

Im OOSE soll die Systemgrenze durch das Modell des Problemgebiets bestimmt werden.

> „ When working with the requirements model it can sometimes be difficult to define the task of the system and especialy the system boundaries. This is typically the case when the requirements specification exists only in a very vague form. Then, a very good tool is to start to develop a logical view of the system using problem domain objects, that is, objects that have a direct counterpart in the application environment and that the system must now it.“ [15, S.167]

Dieses Zitat entwirft viele Fragen im Hinblick auf die Grenzenkonstruktion. Wie sollen z.B. die Methodenanwender/innen entscheiden, wann es schwierig ist, das System und dessen Grenzen zu definieren? Die Antwort auf diese Frage, wenn Anforderungen „ungenau“ definiert sind, ist ungenügend. In diesem Zusammenhang stellt sich eine weitere Frage, wann eine Anforderungsspezifikation „ungenau“ bzw. „genau“ betrachtet werden sollte.

Die Wichtigkeit des mentalen Konstrukts bei der Grenzenkonstruktion drückt in [17, S.76] Jayaratna folgendermaßen aus:

> „We believe that if we, as the intended problemsolvers, do not subject our ´mental constructs´ to self critical examination, then we may indeed end up accepting the boundaries as defined by our clients."

Im OOSE wird zwar beschrieben, daß das Use-Case-Modell und das Modell des Problemgebiets hauptsächlich zum Verstehen des Systems und zum Analysieren der Anforderungen verwendet werden, um mit den Auftraggebern besser kommunizieren zu können. Es wird bei der Erstellung dieser Modelle jedoch nicht die Wichtigkeit des mentalen Konstrukts betont. An dieser Stelle muß darauf hingewiesen werden, daß auch diese beiden Modell an sich den Methodenanwendern beim Verstehen der maßgeblichen Situation unterstützen, diese aber im Sinne von Jayaratna nicht genügend sind, weil sie dabei die Subjektivität der Methodenanwendenden nicht berücksichtigen.

Folgendes Zitat bestätigt, daß bei der Erstellung der Modelle im OOSE keine genügende Hilfe von Seiten der Entwickler geleistet werden.

> „Rules und criteria for how the work should be carried out to yield a good analysis, a good design and so on, usually take no more than a few pages."[15, S.504]

4.3.2. Eine Diagnose durchführen (wo sind wir jetzt?)

Mit Hilfe der Diagnose sollen, Lücken und Mißverständnisse geklärt werden können. Weiterhin dient sie sowohl als Kommunikationsgrundlage mit den Auftraggebern als auch als Basis für weitere Aktivitäten. Dabei muß nach Jayaratna auch der Einfluß des mentalen Konstrukts berücksichtigt werden.

Nun ist die Frage zu beantworten, welche Modelle oder Techniken im OOSE angeboten werden, um die Situation zu beschreiben.

Im OOSE stellt das Use-Case-Modell als eine Art Diagnose dar. Dieses Modell soll nach den Entwickler der Methode das Verständnis der Situation zum Ausdruck bringen und die Lücken und Mißverständnisse mit den Beteiligten zu klären, helfen. Sie soll auch als Basis für die darauffolgenden Aktivitäten dienen.

Jacobson et al. beschreiben das Use-Case-Modell folgendermaßen:

> „This model is easily to understand and formulate from the user perspective, so we can easily talk to the users and see if we are building the correct system according to their requirements." [15, S.129]

Damit macht die OOSE ihren Anwendern implizit auf die Partizipation der Benutzer im Entwicklungsprozeß aufmerksam. Die Schwäche der Methode in diesem Zusammenhang ist, daß die

Anforderungen bei der Erstellung dieses Modells nur aus der Perspektive des Benutzers definiert werden. Es müßte dabei jedoch auch die Perspektive der Methodenanwendenden berücksichtigt werden. Die Wichtigkeit des kritische Hinterfragungs der Gründe für die Anforderungen wird im OOSE hiermit nicht betont. Hierbei ist außerdem zu berücksichtigen, daß aufgrund des spezifischen mentalen Konstruktes, die Einzelnen im Softwareentwicklungsprozesses die Realität unterschiedlich wahrnehmen. Aus diesem Grunde sollte bei der Erstellung des Use-Case-Modells, die Beschreibung der Situation nicht nur von der Perspektive der Benutzer gesehen, sondern soll auch eigene Perspektive berücksichtigt werden. Das obige Zitat aber deutet darauf hin, daß die Korrektheit dieses Modells von den Benutzern bestimmt wird, was aber nach Jayaratna auch ein Kritikpunkt darstellt:

> „(...) the diagramms helped to establish the relevance of information rather than their ´correctness´. Correctness is possible only (...) proof of demonstrable consistent characteristics, i.e. the design of logic circuits, operations of a chemical process, etc.“ [17, S.143]

Jayaratna deutet in diesem Zusammenhang auch darauf hin, daß die Personen dazu neigen könnten, die Modelle als korrekt zu bezeichnen, eventuell aus der Angst heraus, daß durch eine Nichtzustimmung ihrerseits, es zu Problemen bzw. Differenzen innerhalb der Organisation kommen könnte. Daraus kann man dann folgern, daß die Methodenanwender/innen bei der Erstellung des Use-Case-Modelles, zum einen auf die kritische Hinterfragung wert legen müssen und zum anderen die Korrektheit nicht ausschließlich von den Benutzern bestimmen zu lassen, sondern auch ihre eigene Bewertung berücksichtigen sollen.

4.3.3. Konturen der Prognose erstellen (wo wollen wir hin und warum?)

Hier wird untersucht, ob die OOSE eine Hilfe für die Beschreibung des gewünschten Zustandes bietet und wie diese Wünsche von den Auftraggebern übernommen werden.

Die Hinterfragung der Gründe für die Wünsche der Benutzer steht im NIMSAD im Vordergrund. Jayaratna betont in diesem Zusammenhang, daß viele Methoden diese Hinterfragung nicht fordern, sondern ihre Anwender dazu verleiten die Wünsche der Auftraggeber ohne kritische Untersuchung zu übernehmen.

Er sagt dazu:

> „Naturally, it is assumed that the clients and problem owners have a more thorough knowledge of the ´action world´ and have good reasons for their expectations. (...) we as intended solvers should examine the validity of their expectations. (...) the clients and problem owners may or may not have formulated their expactations in a rigorous way.“ [17, S.85]

Jacobson et al. erkennen zwar, daß es bei der Anforderungsermittlung Probleme geben kann:

> „A classic problem in all forms product devolopment is that the orderer and user are different parties. This will lead to conflict about the aim, as orderers and users seldom come up with the same orderer requirements.“ [15, S.18]

Bei der Lösung dieses Problemes jedoch gehen sie aber nur auf die Partizipation der Benutzer und Auftraggeber bei der Anforderungsermittlung und nicht wie von Jayaratna erwartet, auf die kritische Hinterfragung dieser Anforderungen bzw. Wünsche der Beteiligten im Software-Entwicklungsprozeß.

> „The conflict can be diminished, either by making the orderer understand and formulate the users requirements or having representatives of the users take part in formulating the requirements.“ [ebd. S.18]

Es stellt sich aufgrund dieses Zitates auch die Frage, inwieweit die von OOSE befürwortete Partizipation die richtige Art dafür ist. Wenn ein Auftraggeber die Wünsche der Benutzer formulieren darf, ist es fragwürdig, ob auch die wirklichen Wünsche der Benutzer dadurch definiert werden. Dieses ist eine falsche Annahme von Seiten der Entwickler der Methode OOSE, da aufgrund des mentalen Konstruks des Einzelnen, nicht davon ausgegangen werden kann, daß die Auftraggeber die Wünsche der Benutzer so wahrnehmen, wie die Benutzer selbst es tun würden.

Innerhalb der OOSE wird das Konzept des Prototyping benutzt, um die Eigenschaften des beabsichtigten Systems zu demonstrieren. Hierbei wird im OOSE die Korrektheit des Systems durch die Auftraggeber und Benutzer bestimmt. Das ist jedoch nicht ausreichend für die Beschreibung der gewünschten Situation, da die dabei die von den Auftraggebern vorgegebenen Wünsche nicht kritisch untersucht werden.

Eine Überlegung in diesem Zusammenhang wäre, daß eine Art Soll-Use-Cases-Modell erstellt werden könnte, vorausgesetzt die Hinterfragung der Gründe für die Wünsche von Benutzern stehen dabei im Vordergrund.

4.3.4. Hindernisse erkennen

Wie schon im vorigen Abschnitt dargestellt, bietet die OOSE keinen Schritt bzw. Modell um die Prognose zu beschreiben. Es besteht im OOSE vielmehr die Annahme, daß nach der Erstellung des Use-Case-Modelles sowie Formulierung der Wünsche seitens der Auftraggeber, unmittelbar mit der Entwicklung des Systems begonnen werden kann. Da aber bei der Ermittlung dieser Wünsche die Gründe dafür nicht verstanden werden, kann die OOSE auch nicht dabei helfen, die Hindernisse zu erkennen, um von der gegenwärtigen Situation zur gewünschten Situation zu gelangen.

Eine Hilfestellung bei der Erkennung von Hindernissen könnte die „change analysis“ in der Unternehmensentwicklung[8] bieten. Da aber sie im OOSE nicht immer Verwendung findet, kann sie auch nicht als ein Bestandteil der OOSE gesehen und damit auch nicht als eine Hilfe für die Erkennung der Hindernisse betrachtet werden.

4.3.5. Das gedankliche System herleiten

Das gedankliche System im NIMSAD ist dasjenige System, daß bei der Transformation der maßgeblichen Situation zur gewünschten Situation hilft. Jayaratna definiert in diesem Zusammenhang die Beschreibung der Merkmale und das zu erwartende Verhalten des Systems als Systemspezifikation oder Anforderungsspezifikation.

Im OOSE wird von der NIMSAD Perspektive her, kein gedankliches System hergeleitet. Die Anforderungsspezifikation wird zwar mit den Benutzern gemeinsam entwickelt, aber nur mit dem Ziel, die Arbeit bei der Entwicklung der darauffolgenden Modelle zu erleichtern sowie die Richtigkeit der Systementwicklung später bestimmen zu können.

> „If we can identify early on all the ways the system will be used and then control development so that the system offers these ways, we will know we are building the right system.“ [16, S.2]

Das gedankliche System soll aber im NIMSAD darstellen, wie die Hindernisse (wenn sie gefunden worden sind) zu lösen bzw. bewältigen sind. OOSE benutzt von der NIMSAD-Perspektive her einen ontologischen Systembegriff, da durch den Einsatz dieser Methode ein „richtiges System“ definieren versucht wird. Jayaratna befürwortet hingeeegen einen epistemologischen Systembegriff, der versucht „relevante Systeme“ zu definieren. Den Grund für den epistemologischen Systembegriff beschreibt Jayaratna folgendermaßen:

> If we replace the ontological use of the term „system“ as employed within the definition with ist use in an epistemological sense of the term, then there cannot be just *one* System but many possible „systems“ whose boundary construction has to be very much a conscious intellectual activity.“ [17, S.94]

4.3.6. Ein konzeptuelles/ logisches Design erstellen

Das Analysemodell im OOSE stellt das konzeptuelle/logische Modell dar, das im NIMSAD Prognose Modell 1 (PM1) genannt wird. Hier ist jedoch hevorzuheben, daß im NIMSAD das konzeptuelle/logische Modell in Verbindung mit dem gedanklichen System erstellt wird. Da im OOSE

[8] Siehe Kapitel 2.

kein gedankliches System hergeleitet wird, kann hier das konzeptuelle/logische Modell nur für das von den Auftraggebern vorgegebene System erstellt werden.

Das Analyse Modell stellt das Prognose Modell 1 (PM1) im NIMSAD dar. Dieses Modell soll eine applikationsorientierte Spezifikation darstellen. Das Ziel dabei ist eine „gute Struktur" für das System zu finden, die robust gegenüber Veränderungen ist und in klare, verständliche Teile geteilt werden kann. Das Analyse-Modell wird im OOSE aus dem Use-Case-Modell abgeleitet. Die Funktionalität des Gesamtsystems wird auf die drei Analyse-Objekttypen (entity usw) verteilt. Es sollen dabei alle logischen Elemente, die sich im System befinden spezifiziert, die Verbindungen und wie diese Elemente gruppiert werden sollen, beschrieben werden. Hierbei werden die Interaktionsdiagramme zur Identifikation der Operationen und das Subsystem-Konzept zur Gruppierung der Objekte zu verwaltbaren Einheiten benutzt.

Alle dieser Techniken bieten eine gute Unterstützung bei der Erstellung des konzeptuellen/ logischen Models für die Methodenanwendern. In diesem Zusammenhang meinen der Entwickler von OOSE:

> „Giving devolopers the architecture of OOSE allows them to use their creativity to develop stable and robust systems „ [15, S.143]

Ob dies wirklich so ist, bleibt zweifelhaft, da der Anspruch der OOSE-Methode ist, aus dem Software-Entwicklungsprozeß einen rationalen, industriellen Prozeß zu machen, der die Ergebnisse des Prozesses menschenunabhängig betrachtet, bleibt zweifelhaft.

4.3.7. Ein technisches Design erstellen

Das technische Design (physical design) stellt im NIMSAD die Überlegung und die Auswahl von Wegen und Mitteln für die Realisierung des logischen Designs dar. In diesem Abschnitt wird untersucht, welche Modelle und Techniken der OOSE-Methode für die Erstellung des technischen Designs Verwendung finden können.

Im OOSE stellt das Design-Modell das Prognose Modell 2 (PM2) im NIMSAD dar. Es entwickelt von der NIMSAD-Perspektive her das technische Design. Die Hilfe für die Einschätzung der Struktur des zu entwickelnden Systems ist eine Unterstützung beim Design. Dieses Modell und damit verbundene Konzepte (Blockkonzept, Subsystemkonzept) und die Diagramme (Interaktionsdiagramme und Zustandsdiagramme) stellen ohne Zweifel eine Hilfe für die Erstellung des Designs dar. Ein Kritikpunkt in diesem Zusammenhang ist, daß die Entwickler der OOSE den Menschenfaktor auch hier außer Acht lassen.

.

„ The people and organisation involved in the development could also affect the design. (...) We will not discuss these issues in this book, but the principal strategy sould be that such faktors sould not affect the system structure.“ [15, S.211]

Das Ziel der Entwickler von OOSE ist es einen vereinheitlichen (unified process) Software - Entwicklungsprozeß zu kreieren, dessen Prozeßablauf möglichst menschenunabhängig durchgeführt werden kann. Aufgrund dieser Zielformulierung wird deutlich, daß die Entwickler der Methode die Beeinflussung des Designs durch die Menschen als einen Störungsfaktor betrachten und versuchen diesen zu beseitigen.

4.3.8. Implementation

Im OOSE wird hierfür das Implementationsmodell innerhalb des Konstruktionsprozesses erstellt. Dieses Modell soll, den zu schreibenden Code beinhalten. Es wird dabei von den Entwicklern der Methode behauptet, daß durch die Erstellung des Design-Modells die Implementationsarbeit im wesentlichen reduziert werden kann und daß dies auch automatisiert werden könnte aber trotzdem Menschen hierfür gebraucht werden.

„Normally, however, we will still need human beings to make this final trasition to source code [15, S.244]

Im NIMSAD soll aber die Implementation die Gültigkeit der vorher erstellten beiden Modelle zeigen. Dieses kann im OOSE nicht mit dem Implementationsmodell gezeigt werden, sondern lediglich mit dem Testmodell bzw. im Testprozeß.

„(...) whenever a design at one stage has been completed, the testing for that stage should be done at that stage. This means that design, implementation and testing are very interleaved aktivities performed in an incremental fashion.“ [17, S.318]

Das Testen kann mit anderen Aktivitäten parallel eingesetzt werden und somit kann auch die Gültigkeit der beiden Designmodelle geprüft werden.

Jayaratna betont außerdem, daß in diesem Schritt die Zusammenarbeit zwischen Methodenanwender/innen und anderen Beteiligten im Software- Entwicklungsprozeß (Auftraggeber, Benutzer usw.) sehr wichtig ist, um die Implementation erfolgreich durchzuführen. Das ist im OOSE durch die Aktivität Validation im Testprozeß gewährleistet.

Validation is mainly captured by a thorough requierements analysis including active involvement of customers, use of prototypes and so on.“ [15, S.313]

4.4. Evaluation

Die Entwickler der OOSE sehen keinen Schritt zur Evaluation im Sinne von NIMSAD in ihrer Methode vor.
Die Wichtigkeit des Lernens von der Evaluation drückt Jayaratna folgendermaßen aus:

> „Intervention in the „action world“ is a complex activitiy because it affects not only the usually functional and technical aspects of organisations, but also the psychological, political and emotional aspects of their members. Learning, therefore, is an extremely important and fundemental development process. [17, S.215]

Im OOSE wird durch Testen die Korrektheit des entwickelten Systems evaluiert. Allerdings wird dabei nicht auf die im obigen Zitat von Jayaratna genannten Aspekte eingegangen.
Jacobson et.al. stellen zwar die Wichtigkeit des Lernens von den früheren Erfahrungen für den späteren Erfolg der Aktivitäten, das allerdings aber nur im Zusammenhang mit dem Testen und zur Kostensenkung. Nicht hingegen, um aus dem Entwicklungsprozeß und vom Einsatz der Methode zu lernen.

> „The experiences of the testing activity are collected and discussed in order to learn for future test activities. Since experience is always highly coveted and always costs so much, it is important to make use of it.“ [15, S.338]

Aufgrund des Anspruchs der Methode OOSE, den Software- Entwicklungsprozeß zu industrialisieren, ergibt sich die Annahme, daß im OOSE keine Evaluation im weiten Sinne durchzuführen braucht. Daher ist die OOSE auch nicht in der Lage die Methodenanwender/innen zum einen auf die Wichtigkeit des Lernens und zum anderen auf die Evaluation aufmerksam zu machen.

5. Ergebnisse der Studienarbeit

5.1. Bewertung des NIMSAD-Frameworks

Wie diese Studienarbeit auch zeigt, kann das NIMSAD Framework für die Evaluation der OOSE-Methode angewendet werden. Das NIMSAD Framework eignet sich meine Ansicht nach sehr gut als Leitfaden für das Verstehen und für die Beurteilung der Methoden. Für die Softwareentwickler/innen ist es sehr hilfreich, dieses Framework und den Ansatz von Jayaratna nämlich, daß die Methodenanwendung nicht nur die Durchführung der Methodenschritte bedeutet, sondern das sie vielmehr ein kontinuierlicher Lernprozeß ist, zu kennen.

Im Rahmen meiner Untersuchungen habe ich feststellt, daß dem Thema Evaluation in der Literatur wenig Aufmerksamkeit gewidmet wird. Es wird vielmehr versucht, die Methoden mit ihren Modellen und Konzepten zu vergleichen und sie aufgrund ihrer quantitativen Merkmale zu bewerten. Eine kritische Hinterfragung, warum diese Modelle in der Methode enthalten sind und ob sie auch das, was von der Methode behauptet wird, leisten, erfolgt dabei nicht. Hierbei wird immer wieder vergessen, die Subjektivität der Methodenanwender/innen auch bei der Bewertung bzw. Vergleich in Betracht zu ziehen. Das NIMSAD-Framework nach meiner Auffassung ist in dieser Hinsicht gut zu bewerten, weil es auch diesen Aspekt bei der Bewertung von Methoden betrachtet.

Der Ansatz von Jayaratna könnte für die Wissenschaft als Diskussionsbasis oder Anregung hilfreich sein. Es ist meines Erachtens jedoch fraglich, ob dieses Framework auch in der Praxis eine weite Verbreitung finden kann. Da aufgrund der Machtstrukturen innerhalb der Unternehmen und vielen anderen Organisationen die eigentlichen Methodenanwender/innen oftmals gezwungen werden, die von der oberen Hierarchieebene vorgegebenen Methoden, die sich meistens am Verbreitungsgrad einer Methode orientieren, anwenden müssen.

5.2. Schlußfolgerungen aus der Evaluation der OOSE-Methode

Die Evaluation der OOSE Methode mit dem NIMSAD-Framework zeigte, daß die OOSE-Methode obwohl sie nützliche Techniken (z.B. Interaktionsdiagramme) und Konzepte (z.B.Use-Case-Konzept) für die Beschreibung und Entwerfung für Softwaresysteme bietet, jedoch aufgrund ihres Anspruches den Software-Entwicklungsprozeß zu industrialisieren und rationalisieren, viele Aspekte dabei ausklammert (wie z.B. den Methodenanwendenden und auch die Evaluation dieses Prozesses).

Aus diesem Grund ist es meines Erachtens bei der Anwendung der OOSE-Methode wichtig, über diesen Anspruch bewußt zu werden, indem die Methodenanwender/innen versuchen, diese entweder durch Auswahl einer anderen Methode oder durch Kombination mit einer anderen Methode, die diese Aspekte berücksichtigt, zu überwinden. Es ist hierbei auch wichtig, daß die Anwender/innen beim Einsatz dieser Methode in der jeweiligen Situation ihr eigenes mentales Konstrukt und das von anderen in Betracht ziehen, mit dem Ziel aus dem Prozeß der Softwareentwicklung zu lernen und daraus Schlußfolgerungen ziehen zu können.

Literaturverzeichnis

1. /Balzert96/
   Balzert, Helmut: *Lehrbuch der Software-Technik: Software-Entwicklung*, Heidelberg : Spektrum, Akademischer Verlag, 1996

2. /Balzert,Heide96/
   Balzert, Heide: *Objektorientierte Systemanalyse: Konzepte, Methoden, Beispiele*, Heidelberg: Spektrum Akademischer Verlag, 1996

3. /Bittner,Hesse,Schnath92/
   Bittner, U., Hesse, W., Schnath, J.: *Untersuchungen zum Methodeneinsatz in Software Entwicklungsprojekten*, in Softwaretechnik-Trends 1992, Band 12, Heft 3, Seiten 48-60

4. /Denert93/
   Denert,E.: *Software-Engineering in Wissenschaft und Wirtschaft: Wie breit ist die Kluft*?, in Informatik Spektrum 1993, Band 16, seiten 295-299

5. /Duden Informatik93/
   Duden Informatik: *Ein Sachlexikon für Studium und Praxis* (2. Aufl.) Mannheim: Duden Verlag, 1993

6. /Endres93/
   Endres,A: *Software und Software-Entwicklung im Wandel: ein historischer Vergleich,* in Informatik-Spektrum 1993, Band 16, Seiten 261-267

7. /Floyd84/
   Floyd,C: *Eine Untersuchung von Software-Entwicklungsmethoden*. In:
   Morgenbrod, H., Sammer, W.:Programmierumgebungen und Compiler. Stuttgart: Teubner

8. /Floyd86/
   Floyd,C.: *A Comparative Evaluation of System Devolopment Methods*, in Information System Methodologies

9. /Floyd93/
   Arbeitsunterlagen zur Lehrveranstaltung :Einführung in die Softwaretechnik: Universität Hamburg, Sommersemester 1993

10. /Floyd94/
    Floyd,Christiane: *Software-Engineering – und dann*? In Informatik-Spektrum, 1994, Seiten 29-37

11. /Floyd,Züllighoven97/
    Floyd, C., Züllighoven,H.: in Rechenberger, Pomberger: Informatik Handbuch, München: Hanser Verlag,1997, Seiten 641-667

12. /Hasselbring96/
    Hasselbring, W.: *Erfahrungen mit dem Einsatz von Objectory für vorlesungs-begleitende Übungen in der Softwaretechnik-Ausbildung*, in Softwaretechnik-Trends 1996, Band 16, Heft 2, Seiten 10-16

13. /Hesse97/
    Hesse,W.: *Wie evolutionär sind die objektorientierten Analysemethoden? Ein kritischer Vergleich*, in Informatik Spektrum 1997, Band 20, Seiten 21-29

14. /Hügli, Lübcke91/
    Hügli, A., Lübcke, P. (Hg.): *Philosophielexikon*. Reinbek:Rowohlt, 1991

15. /Jacobson92/
    Jacobson,I., Cristersson M., Jonsson P.,Övergaard G.,: *Object-Oriented Software Engineering- A Use Case Driven Approach*, Addison-Wesley 1992

16. /Jacobson94/
Jacobson, I.: *Basic Use-Case Modeling*, www.unantes-univ-nantes.fr /usecase/root.htm, 1994, April 1998

17. /Jayaratna94/
Jayaratna, N.: *Understanding and Evaluating Methodologies, NIMSAD: A systemic Framework*, London: McGraw-Hill 1994

18 /Klaeren94/
Klaeren, H.: *Probleme des Software-Engineering,* in Informatik-Spektrum, 1994, Band 17, Seiten 21-28

19. /Klischewski96/
Klischewski, R.: *Anarchie – ein Leitbild für die Informatik: von den Grundlagen der Beherrschbarkeit zur selbstbestimmten Systementwicklung*, Frankfurt, Europäische Hochschulschriften: Reihe 41, Informatik; Bd. 24

20. /Ott94/
Ott, H., J.: *Das „Ingenieurmäßige" am Software Engineering*, in Softwaretechnik-Trends 1994, Band 14, Heft 1, Seiten 31-38

21. /Pomberger, Blaschek93/
Pomberger, G., Blaschek, G.: *Software Engineering: Prototyping und objektorientierte Software-Entwicklung*, München: Hanser 1993

22. /Raasch92/
Raasch, J.: *Systementwicklung mit strukturierten Methoden: ein Leitfaden für Praxis und Studium*, 2. Auflage, München: Hanser 1992

23. /Reisen89/
Reisen, F., M.: *Gestaltbarkeit und Gestaltung von Methoden – zwei notwendige Bedingungen kooperativer Softwarentwicklung*, in Softwaretechnik-Trends 1989, Band 9, Heft 2, Seiten 14-27

24. /Schmidt82/
Schmidt, Heinrich: *Philosophisches Wörterbuch*, 21.Auflage, Stuttgart 1982

25. /Stein94/
Stein, W.: *Objektorientierte Analysemethoden:Vergleich, Bewertung, Auswah*l. Mannheim: BI-Wissenschaftsverlag, 1994

26. /Stein93/
Stein, W.: *Objektorientierter Analysemethoden-ein Vergleich*, in Informatik Spektrum 1993, Band 16, Seiten 317-333

27. /Weiß97/
Weiß, C.: *Das Use-Case-Konzept*, in Softwaretechnik-Trends 17, November 1997, Seiten 26-31