Moderne Analyse-Methoden

Analyse-Werkzeuge sind aus der modernen Softwareentwicklung nicht mehr wegzudenken. Mit diesem Beitrag möchten wir Interessierten einen Einblick in die gängigen Methoden geben. Die kurze Zusammenstellung soll dabei helfen, das passende Werkzeug auszuwählen.

Wenn Sie diesen Beitrag aufmerksam lesen, werden Sie eine Vorstellung davon haben, was die Begriffe "strukturierte Analyse", "Real Time Analysis" und "objektorientierte Analyse" bedeuten. Sie werden die Vor- und Nachteile der einzelnen Methoden kennen und zumindest grob abschätzen können, für welche Fälle Sie welches Werkzeug einsetzen. Für weiter gehendes Interesse geben wir im Anhang eine ausführliche Literaturliste an.

1. Der Analyse-Begriff

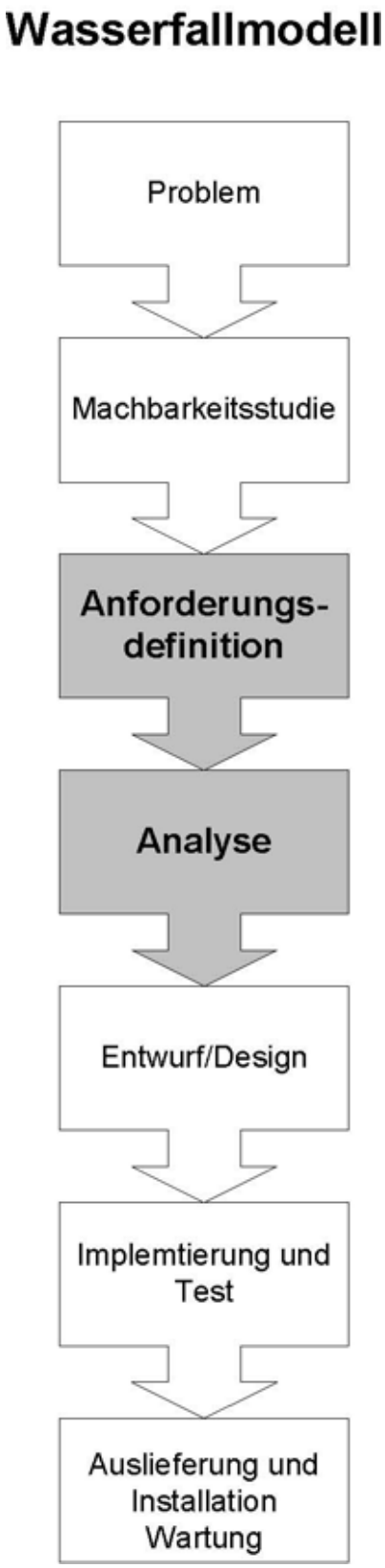
Bevor wir uns in das Thema stürzen, möchten wir den Begriff der Analyse definieren und abgrenzen. Nach unserer Auffassung gelten folgende Definitionen:

• *Analyse* ist ein Prozess, der von der Wahrnehmung der realen Welt zu einem Bild (Darstellung, Modell) dieser Wahrnehmung führt. Bei der Analyse machen wir uns ein idealisiertes Modell der realen Welt, ohne auf die diversen Randbedingungen oder Einschränkungen einer möglichen Implementierung einzugehen (Beispiel: EDV System mit unendlich großem Speicher und unendlich großer Prozessorleistung).

• *Design* hingegen ist der Prozess, der beschreibt, wie aus der oben gewonnenen Darstellung eine Implementierung erreicht werden kann. Beim Design müssen wir das in der Analysephase entwickelte Modell an die Realität anpassen, um eine Realisierung (=Implementierung) zu ermöglichen (bezogen auf obiges Beispiel bedeutet das, nun von realen EDV Systemen auszugehen).

Dazu noch ein einfaches Beispiel am Konzept eines "bewohnbaren Objekts": Die Analyse der Anforderungen könnte darauf führen, dass für eine Familie eine passende Behausung gesucht wird, die unendlich groß ist, über einen unbegrenzten Garten verfügt, perfekte Verkehrslage aufweist und in einer höchst gesunden Gegend liegt. Das Design würde zu genaueren Spezifikationen innerhalb der realen (wie etwa finanziellen) Rahmenbedingungen führen. Ergebnis könnte sein, dass ein zweistöckiges Haus mit Solarzellen, drei Schlafzimmern, zwei Bädern, einer Küche, einem Esszimmer, einem Wohnzimmer, einem Garten mit viel Grün und einem Pool sowie einer Garage ca. 30 km außerhalb von Wien gesucht wird.

Die Abgrenzung von Analyse und Design ist deshalb wichtig, da die Begriffe in der Literatur gerne vermischt werden bzw. der eine für das andere verwendet wird. Zur weiteren Veranschaulichung unserer Abgrenzung sind die Zusammenhänge auch in dem Wasserfall-Modell der Softwareentwicklung ersichtlich, das auf der Folgeseite abgebildet ist.



Das Wasserfall-Modell der Softwareentwicklung: Es zeigt u.a. die Abgrenzung von Analyse- und Designphase.

2. Strukturierte Analyse (SA)

Die Methode der strukturierten Analyse (SA) wurde 1978 von Tom de Marco beschrieben (Structured Analysis and System Specification). Im Laufe der Zeit wurde sowohl die Methode als auch die Notation modifiziert und weiterentwickelt. SA gilt in der IT-Industrie immer noch als state-of-the-art.

SA besteht aus einem Hierarchiemodell, das die einzelnen Datenflussdiagramme als Baum anordnet. Wurzel des Baumes ist das Kontextdiagramm. Blätter sind die Datenflussdiagramme, die nicht weiter verfeinert sind. Prozesse dieser Datenflussdiagramme werden durch so genannte MiniSpecs beschrieben. Innerhalb eines SA-Modells muss die Datenintegrität sichergestellt sein (Man spricht von "balancing" und meint damit, dass die Datenflüsse zwischen Kind- und Elterndiagramm ausbalanciert sein müssen).

Mit der SA geht man von einem abstrakten Datenflussdiagramm (DFD) aus. Jeder Prozess wird verfeinert und durch ein eigenes DFD beschreiben. Wenn ein Prozess nicht mehr unterteilt werden kann (ein Blatt der Hierarchie ist), wird er durch eine Mini-Spezifikation näher beschreiben. Mini-Spezifikationen sind Pseudocode, Entscheidungstabellen oder Entscheidungsbäume.

In der SA *verwendete Basiskonzepte* sind: Data Dictionary, Datenflussdiagramme, Entscheidungstabellen, Pseudocode und Funktionsbäume.

Vorteile von SA:
- geschickte Kombination bewährter Basiskonzepte
- hierarchisch gegliederte Datenflussdiagramme führen zur Verbesserung der Übersichtlichkeit
- leicht erlernbar
- gut nachvollziehbar
- viele CASE Werkzeuge unterstützen SA
- erlaubt top-down Einarbeitung in System
- guter Zusammenhang zu Entity Relationship Modellen über Speicher darstellbar

Nachteile von SA:
- keine Verfeinerung von Schnittstellen möglich
- Strukturbrüche bei der Transformation eines SA-Modells in einen daten-abstraktionsorientierten Entwurf.
- keine Möglichkeit, Speicher zu verfeinern

Zugehörige Designmethoden siehe:

Yourdon/Constantine: Structured Design
Englewood Cliffs: Prentice Hall 1979

Page-Jones: The Practical Guide to Structured Design
Englewood Cliffs: Prentice Hall 1988

3. Real Time Analysis (SA/RT)

SA/RT erweitert die SA zur Modellierung von ereignisgesteuerten Systemen mit Zeitanforderungen und komplexen Prozessaktivierungen, da SA es nicht erlaubt, Zeitanforderungen, Terminierung und Initialisierung von Systemen zu beschreiben. SA/RT ist Stand der Technik für technische Systeme.

Real Time Analysis erweitert SA um die Möglichkeit, Prozesse zu aktivieren und zu deaktivieren. Es können zusätzliche Zeitspezifikationen beschrieben werden, für die es neben Datenflüssen auch Kontrollflüsse gibt, die für Ereignisse stehen. Datenflussdiagramme (DFD) werden zu FD (Flussdiagrammen) verallgemeinert. Diese enthalten auch Kontrollflüsse. Durch so genannte Kontrollflussspezifikationen (Cspec) erfolgt die Steuerung der Prozesse, die sich auf einem FD befinden. In Kontrollflussspezifikationen verwendete Input/Output Signale werden durch Balken-Notation im dazugehörenden FD angegeben. Elementare Prozesse beschreibt man durch Prozessspezifikationen (Psecs). Flüsse und Speicher werden in einem Requirements Dictionary definiert.

Vorteile von SA/RT:
• sehr gute Eignung zur Modellierung ereignisgesteuerter Systeme
• komplexe Steuerungszusammenhänge können beschrieben werden
• zurzeit state-of-the-art für diese Anwendungsklasse
• guter Support durch Tools

Nachteile von SA/RT:
• schwieriger zu verstehen als SA
• manchmal eher unübersichtlich, vor allem bei großen CSpecs
• Verletzung des Lokalitätsprinzips möglich
• Dynamik schwer nachzuvollziehen

Zugehörige Designmethoden siehe:
Ward/Mellor: Structured Development for Real-Time-Systems
Englewood Cliffs: Prentice Hall 1985

Beispiele für *strukturierte Programmiersprachen*:
(im Grunde alles, was kein "GOTO" braucht)
C, Pascal, Fortran (mit Einschränkungen) und Basic (mit Einschränkungen)

4. Objektorientierte Analyse (OOA)

Die hier beschriebenen traditionellen Analysemethoden wie SA und SA/RT bringen eine Reihe von Problemen mit sich. Dazu gehören zum Beispiel:

• Künstliche Trennung von Daten und Funktionen
• Verschiedene Modelle für Analyse und Design
• Instabilität der Architektur
• Skalierung schwierig
• Vernachlässigung der Wiederverwendbarkeit
• Änderungsempfindlichkeit der Implementation
• kein Support für Datenabstraktion
• kein Support für Information-Hiding

Diese und andere Probleme führten zur Suche nach neuen Möglichkeiten. Die Hoffnung war, mit Objektorientierung eine bessere Wiederverwendbarkeit und damit geringere Entwicklungskosten bei der Softwareproduktion zu erreichen. Reduzierung von Integrationsproblemen, Verringerung des Unterschieds zwischen Analyse und Design sowie konzeptuelle Integrität sind weitere Punkte, die bei der Anwendung von OO Techniken Vorteile bringen sollen. Schon Anfang der 80er gab es die erste OO Programmiersprache (Smalltalk). Ende der 80er war die Zeit reif für den Aufbruch des objektorientierten Ansatzes. Anfang der 90 Jahre wurden an die 50 verschiedenen Methoden der OO Analyse beschrieben. Zuerst wurden OO Analyse Methoden beschrieben, dann OO Design und schließlich setzen sich OO Sprachen (C++, Java) durch. So betont man zum Beispiel an der Softwareschule der technischen Universität Bern: "Objektorientierte Analyse und objektorientiertes Design sind heute die zeitgemäßen Ansätze für die Modellierung von komplexen Softwaresystemen. Die Resultate von OOA/OOD lassen sich mit objektorientierten Programmiersprachen wie zum Beispiel C++, Java, Smalltalk oder Eiffel direkt umsetzen, was zu einem durchgehend objektorientierten Entwicklungsprozess führt. Ist dieser Prozess einmal gut eingespielt, sinken die Entwicklungszeiten und damit auch die Kosten der Softwareentwicklung. Systeme, die mit Hilfe von OO Techniken (OOA, OOD, OOP) entwickelt wurden, sind - bei richtiger Anwendung der Techniken - stabiler, kompakter und leichter wartbar als mit funktionsorientierten Methoden entwickelte Systeme. Zudem können Teile der entwickelten Software - mit vertretbarem Aufwand und entsprechenden organisatorischen Maßnahmen - in zukünftigen Projekten wieder verwendet werden. Im Bereich OO Methoden wurde in der Zwischenzeit ein Konsolidierungsprozess durchlaufen, welcher als Resultat die nun für den produktiven Einsatz reife Modellierungssprache UML (Unified Modeling Language) hervorgebracht hat. In der Softwarebranche hat diese bereits hohe Akzeptanz und Unterstützung.“

Vorteile von OOA:
• Wiederverwendbarkeit
• Softwareprojekte unter Verwendung von OO Technologien sind kostengünstiger
• mehrere Schichten auf das zu beschreibende Problem werden integriert
• gute Tools am Markt

Nachteile von OOA:
• relativ kompliziert

5. Methoden der OOA

Seit Anfang der 90er gibt es viele verschiedene Methoden der Objektorientierten Analyse. Hier eine kurze Auswahl:

Shlaer/Mellor 88,92: InfraStrukturdiagramme
Coad/Yourdon 91: OOA Modell
Wirfs-Brock, Wilkerson, Wiener 90: C(lass)R(esponsibility)Collaboration cards
Booch 94: Klassendiagramme
Rumbaugh etc al 91: OMT
Martin/Odell 1992
Ivar Jacobson 92: Analysemodelldiagramm (Use cases)

Grundsätzlich unterscheidet man zwischen:

- *datenorientierten Methoden*: diese gehen erst im 2. Schritt auf dynamische Aspekte des Modells ein. Dazu gehören Rumbaugh und Shlaer/Mellor.
- *verhaltensorientierten Methoden*: wie Wirfs-Brock, Wilkerson & Wiener
- *Synthesen* zwischen beiden Vorgehensweisen: Coad/Yourdon, Booch

Darüber hinaus gibt es noch eine Unmenge weiterer beschriebener Methoden, die zum Teil rein akademischen Charakter haben oder aber auch sehr pragmatische Ansätze verfolgten. Die meisten Methoden beinhalten eine Modellierungssprache und einen Prozess.

In der OOA *verwendete Basiskonzepte* sind: Objekt, Attribut, Operation, Klasse, Botschaft, Vererbung, Polymorphismus, Assoziationen, Aggregationen, Kardinalitäten, Subsysteme und Pattern.

Die Erstellung eines OOA-Modells auf Basis von schriftlichen Grundlagen (Pflichtenheft) bzw. auf Grund von Interviews mit Anwendern ist eine schwierige Aufgabe und gehört zu den großen Herausforderungen der praktischen Softwaretechnik. Da es keine standardisierte Vorgehensweise gibt, beschreiben wir hier kurz eine mögliche Methode: Der Vorgang ist iterativ. Man geht von einem Basismodell aus, das in vielen Schritten verbessert und präzisiert wird. Als Beschreibungssprache sollte man heute grundsätzlich UML (siehe unten) verwenden, das sich als Standard weitgehend durchgesetzt hat.

Start

1. Finden von Klassen
2. Finden von Assoziationen und Aggregationen
3. Finden von Attributen und externen Operationen je Klasse
4. Finden des Objekt - Lebenszyklus
5. Vererbungsstrukturen
6. Finden der internen Operationen für jede Klasse
7. Spezifikation der Operationen
8. Check ob alles ok (Vererbung, Assoziationen etc)
9. Aufsuchen von Subsystemen

Fang oben wieder an

6. Unified Modeling Language (UML)

Mitte der 90er gelang Rational der große Wurf, die Exponenten der drei damals wichtigsten Methoden für ihre Firma zu verpflichten. Booch, Rumbaugh und Jacobson (die so genannten Amigos) wollten Mitte der 90er Jahre eine einzige, gemeinsame Methode entwickeln. Da aber Analyse ein kreativer Prozess ist, kann man für die Vorgehensweise nur Tipps und Hinweise geben, aber keine EINZIGE allgemein gültige Methode. Sehr wohl konnten die Amigos aber ein Werkzeug (Notation) entwickeln, die alle für den Bedarf der Analyse benötigten Begriffe und Modelle enthält. So entstand UML, eine Modellierungssprache, die sehr schnell von der Industrie angenommen wurde und von der OMG (Object Management Group) als Standard definiert wurde.

UML vereinheitlicht im Wesentlichen die Methoden von Booch, Rumbaugh (OMT) und Jacobson, ist aber umfassender als die Summe der drei Methoden.

UML selbst beschreibt kein einheitliches Vorgehen, sondern gibt nur ein Werkzeug in die Hand (Modellierungssprache).

UML bedient sich acht verschiedener Sichten, die in Diagrammen dargestellt werden:
- Use-Case Diagramm
- Sequenzdiagramm
- Zusammenspieldiagramm
- Zustandsdiagramm
- Aktivitätendiagramm
- Komponentendiagramm
- Verteilungsdiagramm
- Klassenstrukturdiagramm

Der OOA *zugehörige objektorientierte Designmethoden* (OOD) sind nicht 100%-ig notwendig aber sehr nützlich für einheitliches Vorgehen im OO-Bereich.
Da die OOA keine technischen Lösungen beschreibt, keine Optimierungen enthält und keine Verwaltung von Objekten anbietet (sowie alle Assoziationen und Aggregationen bidirektional sind) braucht man Methoden, die zum Ziel haben, die in OOA beschrieben Modelle in eine Architektur zu bringen bzw. einen Entwurf für die Implementierung zu unterstützen. Für diesen Zweck wurden diverse Konzepte für den Objektorientierten Entwurf (OOD) entwickelt, deren Besprechung den Rahmen dieses Beitrags sprengen würde. Betreffend weiterer Informationen verweisen wir auf die Literaturliste.

Ergänzend zu diesem Text steht Ihnen auf der InfraSoft Website ein Glossar zur Verfügung, in dem Sie die meisten der hier verwendeten Begriffe finden. Über das aktuelle Angebot an weiteren, kostenlosen Fachbeiträgen zur Softwareentwicklung informieren Sie sich bitte unter www.infrasoft.at/service.

Klaus Rogetzer
Wien, im Mai 2002

Der Autor ist Mitarbeiter der InfraSoft, einem Unternehmen, das auf komplexe Softwareentwicklungen spezialisiert ist. Die Experten der InfraSoft haben langjährige Erfahrungen in der Entwicklung und verfügen über fundierte Kenntnisse in Design, Analyse, Realisierung, Test und Projektmanagement.
Für **individuelle Beratungen** zur Entwicklung von Softwarelösungen und die Bereitstellung von **Realisierungsteams** wenden Sie sich bitte an info@infrasoft.at.

Literatur

Strukturierte Analyse (SA)
Strukturiertes Design (SD)

De Marco T.:
Structured Analysis and System Specifications, Englewood Cliffs: Yourdon Press 1978

Yourdon E.:
Modern Structured Analysis
Englewood Cliffs: Yourdon Press 1989

Page-Jones:
The Practical Guide to Structured Design
Englewood Cliffs: Prentice Hall 1988

Real Time Analysis (SA/RT)

Ward/Mellor:
Structured Development for Real-Time Systems
Volume 1,2,3.
Yourdon Press 1985, 1986.

Hartley/Pirbhai:
Strategies for real-time system specifications.
New York: Dorset House Publishing 1987

Objektorientierte Analyse (OOA)

Shlaer/Mellor:
Object-Oriented Systems Analysis Modeling the World in Data
Englewood Cliffs: Yourdon Press 1988

Coad/Yourdon:
Object-Oriented Analysis
Englewood Cliffs: Yourdon Press 1991

(Deutsch)
Objektorientierte Analyse
München: Prentice Hall 1994

Wirfs-Brock, Wilkerson, Wiener:
Designing Object-Oriented Software
Englewood Cliffs: Prentice Hall 1990

(Deutsch)
Objektorientiertes Software-Design
München: Hanser 1993

Booch:
Object-Oriented Analysis and Design with Applications
Redwood City: Benjamin/Cummings Publishing 1994

(Deutsch)
Objektorientierte Analyse und Design
Bonn, Addison-Wesley 1994

Rumbaugh et al. :
Object-Oriented Modeling and Design
Englewood Cliffs: Prentice Hall 1991

(Deutsch)
Objektorientiertes Modellieren und Entwerfen
München, Hanser 1993

Martin/Odell:
Object-Oriented Analysis and Design
Englewood Cliffs: Prentice Hall 1992

Jacobson et al. :
Object-Oriented Software Engineering – A Use Case Driven Approach
Wokingham: Addison-Wesley 1992

Objektorientiertes Design (OOD)

Coad/Yourdon:
Object oriented Design
Englewood Cliffs: Yourdon Press 1991

Rumbaugh etc al:
Object-Oriented Modeling and Design
Englewood Cliffs: Prentice Hall 1991

Unified Modeling Language (UML)

Balzert H.:
UML Kompakt,
Spektrum 2001

Burkhardt:
UML – Unified Modeling Language
Objektorientierte Modellierung für die Praxis; Addison Wesley 1997

Fowler, Scott:
UML konzentriert
Addison Wesley 2000

Rumbaugh, Jacobson, Booch
The Unified Language Reference Manual
Addison Wesley 1999

Booch, Rumbaugh, Jacobson
Das UML Benutzerhandbuch
Addison Wesley 1999

Martin/Odell:
Objektorientierte Modellierung mit UML: Das Fundament
Prentice Hall 1999

Softwaretechnik

Helmut Balzert:
Lehrbuch der Software-Technik I,II
Spektrum Verlag

Ian Sommerville:
Softwareengineering, 6th Edition
Addison Wesley 2000

(in deutscher Sprache:)
Softwareengineering, 6. Auflage
Addison Wesley 2001