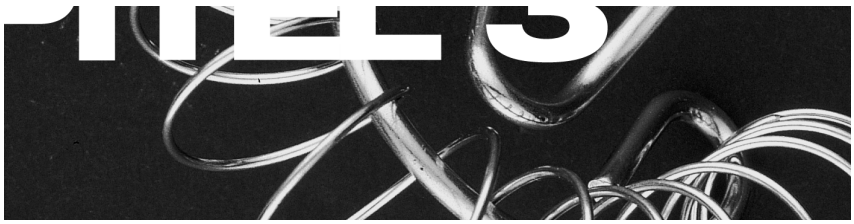
3

Von der Analyse zur Implementierung



3 Von der Analyse zur Implementierung

3.1 Überblick

Dieses Kapitel widmet sich dem Entwicklungsprozess eines Softwareproduktes von der Analyse bis zur Implementierung. Dabei werden Grundprinzipien, Methoden und Techniken vorgestellt, die diesen Prozess unterstützen können. Aus der Vielfalt der vorhandenen Erfahrungen muss sich ein Entwicklungsteam diejenigen heraussuchen, die für das entsprechende Projektvorhaben Erfolg versprechend sind. Einige Methoden und Techniken sind als alternativ zu betrachten, andere können sich aber auch ergänzen.

Zunächst steht die Problematik der Analyse im Mittelpunkt, bevor auf den Entwurf und die Implementation etwas genauer eingegangen wird. Die Anforderungsanalyse ist für den Erfolg oder Misserfolg eines Projektes von ganz entscheidender Bedeutung. Hier werden die Weichen dafür gestellt, was in den folgenden Projektphasen realisiert wird. Anforderungen, die übersehen oder falsch aufgenommen werden, verursachen im späteren Verlauf der Projektentwicklung sehr große Aufwendungen.

Im Verlaufe des gesamten Kapitels erfolgt stets ein Seitenblick auf eine mögliche Werkzeugunterstützung der vorgestellten Methoden, die für die Akzeptanz beim praktischen Einsatz mehr und mehr an Bedeutung gewinnen. Zunächst aber noch einige einführende Bemerkungen zum gesamten Entwicklungsprozess.

Um den Prozess der Softwareentwicklung zu beschreiben, wurden Modelle, die so genannten Lebenszyklusmodelle, erarbeitet.

Definition 3.1 Softwarelebenszyklus

Der Softwarelebenszyklus ist die Menge der einzelnen Tätigkeiten, die während der Prozesse der Entwicklung und Anwendung von Software in einer vorgegebenen Reihenfolge ablaufen und sich technologisch bedingt oder bei veränderten Ausgangsbedingungen zyklisch wiederholen. Er ist in abgegrenzte Teilprozesse, die sogenannten Phasen unterteilt, um den Arbeitsablauf einer Aufgabenstellung arbeitsteilig inhaltlich, technologisch, leitungsmäßig und organisatorisch effektiv zu beherrschen.

Softwareentwicklung ist damit eine Folge von abgegrenzten Phasen. Jede Phase liefert ein abgeschlossenes Ergebnis. Der Übergang von einer Phase zur nächsten beginnt erst nach der Qualitätskontrolle der abzuschließenden Phase.

Der Grundzyklus der Softwareentwicklung lässt sich in folgende Phasen unterteilen:

- **Analysieren**

  Analyse des Basisprozesses

  Ergebnis: Aufgabenstellung zur Softwareentwicklung

- **Spezifizieren**

  Dokumentation der Funktionen des Softwareproduktes

  Ergebnis: funktionelle Spezifikation

- **Entwerfen**

  Dokumentation der Problemlösung

  Ergebnis: logische Gliederung der Funktionen und Daten (fachlicher Entwurf) und Ablaufstruktur (programmtechnischer Entwurf)

  Ergebnis: Entwurfsspezifikation

- **Implementieren**

  Codierung der Problemlösung in einer Programmiersprache, Übersetzen und Verbinden

  Ergebnis: lauffähiges Softwareprodukt

- **Testen**

  Verwendung von vorbereiteten Testmitteln, Testverfahren und Testdaten zum Vergleich von Soll- und Ist-Eigenschaften des Programms – Fehlerfindung

  Ergebnis: Fehlerprotokoll

- **Nutzen**

  Anwendung des Softwareproduktes, Nachweis des stabilen Dauerbetriebs, Nutzereinweisung.

  Ergebnis: laufendes Softwareprodukt

- **Warten**

  Änderung des fertigen Softwareproduktes zur weiteren Nutzbarkeit (Anpassung an neue Bedingungen, Mängelbeseitigung)

  Ergebnis: aktualisiertes Softwareprodukt (samt Spezifikationen).

Als man sich in den sechziger Jahren intensiver mit den Problemen der Entwicklung anwendungsfähiger Software beschäftigte, war man noch der Meinung, dass ein Problem nur lange genug analysiert werden muss, um zu einer vollständigen Anforderungsanalyse zu gelangen. Die Hauptprobleme wurden damals in einer nicht korrekt durchgeführten Analyse gesehen. Man legte das Augenmerk auf verbesserte Spezifikationen in diesem Bereich, ließ aber das Problem der Kommunikation zwischen Entwicklern und Anwendern außer Acht. Erst später erkannte man, dass Formulierungen von Anwendern das eine und ihre wirklichen Vorstellungen etwas anderes sein können. Formal zu beweisen, dass eine Implementation den Anforderungsdokumenten entspricht, reicht für eine erfolgreiche Zusammenarbeit nicht aus. Es bedarf auch einer ständigen Kommunikation mit dem Anwen-

der und einer gleichzeitigen Revision der erarbeiteten Dokumente. Auch die Probleme der Wartung wurden damals noch völlig unterschätzt. Daher war das erste Lebenszyklusmodell zur Beschreibung der Softwareentwicklung noch linear.

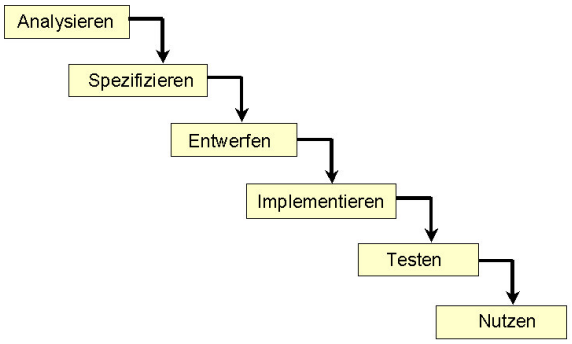


Abbildung 3.1 Klassisches Wasserfallmodell

Das Wasserfallmodell basiert auf den Vorstellungen eines geradlinigen Entwicklungsprozesses von der Analyse bis zur Nutzung. Die Geschichte zeigte jedoch, dass auch Projekte mit einer sehr gründlichen Analyse Probleme bei der Nutzung bekamen. Im Verlaufe der Projektentwicklung ergeben sich einerseits neue Erkenntnisse im Anwendungsgebiet, die bei der Analyse noch nicht berücksichtigt werden konnten. Andererseits entstehen bei der Analyse Kommunikationsprobleme zwischen Anwendern und Entwicklern, die erst zu Tage treten, wenn lauffähige Softwarebausteine vorhanden sind.

Aus diesem Grunde ist eine zyklische, evolutionäre Softwareentwicklung mit Präsentation von Prototypen notwendig. Diese Art der Vorgehensweise gestattet die schnelle Einbeziehung neuer Erkenntnisse und unterstützt ganz wesentlich die Kommunikation zwischen Auftraggebern, Anwendern und Entwicklern.

Die Zustimmung zu lauffähigen Prototypen sichert ein gemeinsames Verständnis des Anwendungsgebietes stärker als statische Spezifikationen in Form von Dokumenten. Das Rapid Prototyping hat sich daher als Erfolg versprechender Ansatz bei der Entwicklung von Software weitgehend durchgesetzt.

Definition 3.2 Rapid Prototyping

Unter dem Rapid Prototyping versteht man das schnelle (rapid) Erstellen eines lauffähigen Systems, das wesentliche Eigenschaften des endgültigen Softwaresystems besitzt.

Oft wird dieser Begriff mit dem partizipativen Prototyping gleichgesetzt, was nicht ganz korrekt ist. Unter partizipativem Prototyping versteht man die Einbeziehung des späteren Anwenders in die Systementwicklung, insbesondere bei der Gestaltung der Benutzungsschnittstelle. Daneben gibt es das explorative Prototyping, bei dem kritische Teilprobleme erkundet werden.

Definition 3.3 Partizipatives Prototyping

Die gemeinsame prototypische Gestaltung der Benutzungsoberfläche des zu entwickelnden Systems zusammen mit dem Anwender bezeichnet man als partizipatives Prototyping.

Definition 3.4 Exploratives Prototyping

Als erkundendes Prototyping bezeichnet man die Implementation von Software, die zur Überprüfung der technischen Machbarkeit kritischer Teile eines Systems dient.

Vor der Entwicklung eines Softwaresystems zum kooperativen Arbeiten wird man beispielsweise erst einmal einen Prototypen erstellen, der ein Byte zwischen zwei Orten überträgt. Von diesen Erfahrungen hängt dann die weitere Gestaltung des Softwaresystems ab. Wenn die Übertragung des einen Bytes bereits sehr zeitaufwändig ist, dann müssen prinzipiell neue Überlegungen angestellt werden.

Das Spiralmodell greift die Idee des Prototyping und des evolutionären Ansatzes auf. Es geht auf Barry W. Boehm /3.1/ zurück und wiederholt zyklisch die Projektabschnitte Planung, Risikoanalyse, Realisierung, Bewertung. Dabei sind auch diese Abschnitte nicht streng getrennt, sondern überlappend.

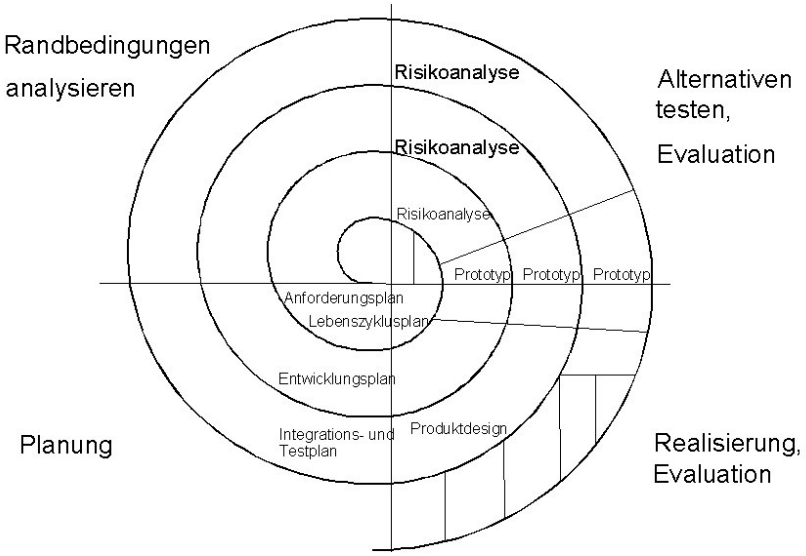


Abbildung 3.2 Spiralmodell nach Boehm

Eine Mischung aus Spiral- und Wasserfallmodell bietet der *Rational Unified Process* an. Verkürzt wird das Prozessmodell mit RUP bezeichnet. Es wurde im Zusammenhang mit

der Entwicklung von UML in der Firma Rational entwickelt /3.22/. Er gliedert die Softwareentwicklung in 4 Phasen, die Konzeptionsphase (inception), die Spezifikations- oder Entwurfsphase (elaboration), die Konstruktionsphase (construction) und die Einführungsphase (transition). Dieses Modell entspricht dem Wasserfallmodell, allerdings wird vorgeschlagen, Spezifikationsphase und Einführungsphase zweimal zu durchlaufen. Für die Konstruktionsphase wird eine mehrfache Wiederholung vorgesehen. Damit wird speziell diese Phase nach einem Spiralmodell abgearbeitet. **Abbildung 3.3** gibt die Aufwandsverteilung für unterschiedliche Aktivitäten nach der RUP-Philosophie an.

RUP ist ein sehr detailliert ausgearbeitetes Prozessmodell für alle Phasen der Softwareentwicklung. Es beschreibt, wer (Rolle) was (Aktivität) unter Nutzung welcher Hilfsmittel (Tool) in welcher Reihenfolge (Workflow) mit welchem Ergebnis (Artefakt) macht.

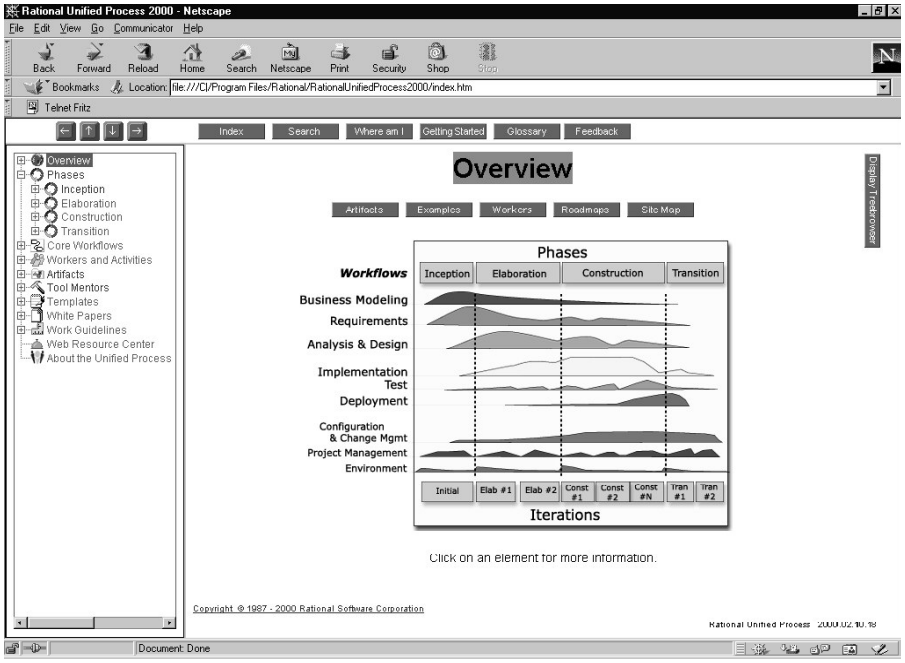


Abbildung 3.3 Aufwandsmodell entsprechend Rational Unified Process

Die Workflowspezifikationen sind in Form von speziellen Notationen von Aktivitätsdiagrammen beschrieben. Zu RUP gibt es eine auf eigene Belange konfigurierbare Werkzeugunterstützung, die den Entwicklern den Prozess genauer erläutert und Templates für Dokumente bereitstellt. **Abbildung 3.4** gibt einen Eindruck von der Präsentation der Informationen über einen Browser.

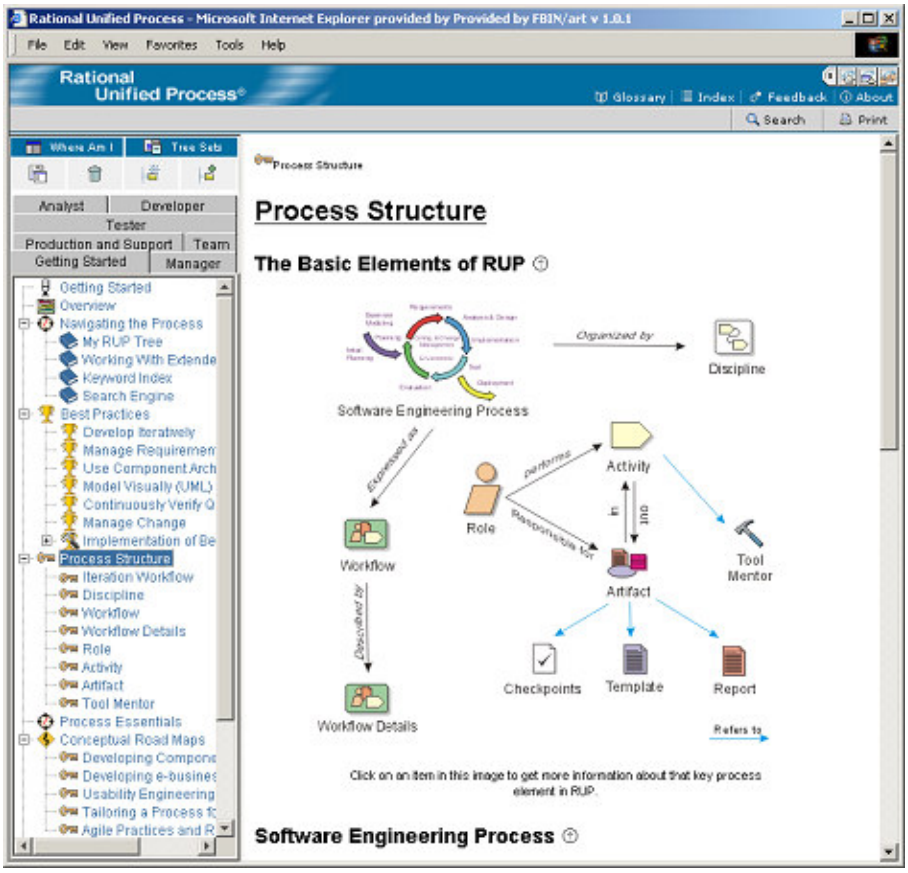


Abbildung 3.4 Screenshot vom Informationssystem zum RUP

In Deutschland ist für öffentliche Aufträge das V-Modell bindend. Ursprünglich entsprach es einer Variante des Wasserfallmodells, wobei der linke Schenkel des V die Projektentwicklung bis zur Implementation repräsentiert. Der rechte Schenkel entspricht den Test- oder Prüfungsaktivitäten der Ergebnisse der Entwicklungsphasen auf dem anderen Schenkel.

Die neue Version des V-Modells unterstützt einen evolutionären Ansatz, der detailliert festlegt, welche Aktivitäten in den einzelnen Phasen durchzuführen sind, wer die Verantwortung dafür hat und welche Werkzeuge genutzt werden. Die Verantwortlichkeiten sind in Form von Rollen definiert. Ein guter Überblick über das V-Modell kann in /3.2/ gefunden werden.

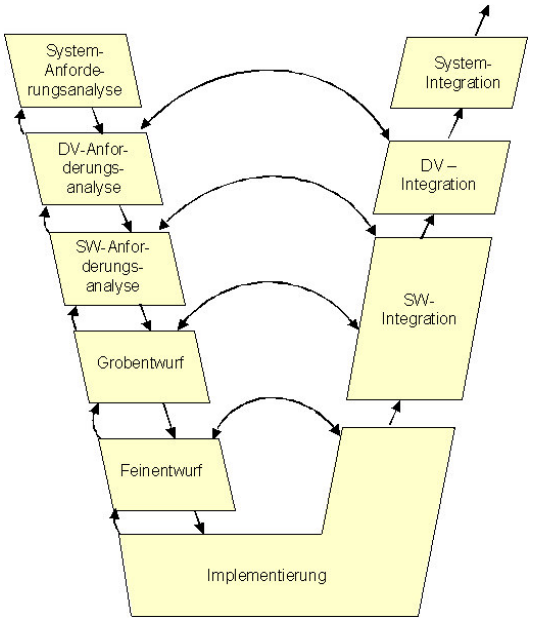


Abbildung 3.5 V-Modell

Ganz entscheidend für den Erfolg von Softwareentwicklern ist die Wiederverwendung von Softwareteilen. Ohne sie sind Projekte auf Dauer nicht mit ökonomischem Erfolg realisierbar. Die Wiederverwendung hat nicht nur den Vorteil, dass die bereits erarbeiteten Ergebnisse erneut genutzt werden können, sondern verringert die Fehlerwahrscheinlichkeit erheblich.

Oftmals müssen Projekte durch ein Reverse-Engineering für eine Wiederverwendung erst zugänglich gemacht werden.

Definition 3.5 Reverse-Engineering

Die Formulierung der Eigenschaften eines vorhandenen Systems auf abstraktem Niveau wird als Reverse-Engineering bezeichnet.

Damit erhält man beispielsweise auf der Basis von vorhandenem Quelltext Spezifikationen in Form von Klassendiagrammen oder Verhaltensspezifikationen, die zum Verständnis hilfreich sind. Diese Spezifikationen können der Ausgangspunkt für die Neugestaltung eines Systems sein, das als Forward-Engineering bezeichnet wird.

Definition 3.6 Forward-Engineering

Die Neugestaltung eines Systems auf der Basis abstrakter Spezifikationen bezeichnet man als Forward-Engineering.

Den gesamten Prozess der Analyse eines vorhandenen Systems und der Neugestaltung auf der Basis der gewonnenen abstrakten Beschreibung bezeichnet man als Re-Engineering.

Definition 3.7 Re-Engineering

Das Reverse-Engineering und das darauf folgende Forward-Engineering werden zusammengenommen als Re-Engineering bezeichnet.

Das Re-Engineering hat immer mehr an Bedeutung gewonnen. Es werden kaum noch neue Anwendungssysteme entwickelt, für die es nicht schon bestehende Programmsysteme gibt. Mit der steigenden Komplexität von Softwaresystemen ist man darauf angewiesen, existierende Software in die Entwicklung einzubeziehen. Es ist nicht mehr möglich, Software immer vollständig neu zu entwickeln.

Für die Erfassung der Anforderungen für Projekte ist die Zusammenarbeit mit den Anwendern unumgänglich. Um festzustellen, ob wirklich die gleichen Vorstellungen von zukünftigen Arbeitsabläufen bestehen, bietet sich die Nutzung von Prototypen der Benutzungsoberfläche an. Diese können mit dem Computer gestaltet sein oder nur einfach skizziert auf Papier vorliegen.

Die Nutzung von Papier hat den Vorteil der Visualisierung der Vorläufigkeit von Ideen. Alle Beteiligten sind mehr geneigt, ihre Vorstellungen einzubringen. Zu perfekte Darstellungen haben etwas den Charakter des Endgültigen und sind eventuell sogar kontraproduktiv für neue Ideen.

Mit Hilfe der Benutzungsoberfläche lassen sich dann Anwendungsszenarien durchspielen, um neue Anforderungen zu ermitteln und um einen Konsens über die bestehenden Vorstellungen von zu entwickelnden Systemen zu erarbeiten. Nutzt man Karteikarten für die Darstellung der Benutzungsoberfläche, so kann analog zur CRC-Karten-Technik eine Sitzung des Entwicklungsteams gestaltet werden. Die Technik wird im folgenden Abschnitt kurz vorgestellt und es wird dem Leser überlassen, die dargestellte Methodik auf Fenster der Benutzungsoberfläche zu adaptieren.

3.2 Analyse

3.2.1 CRC-Karten

Die CRC-Karten-Technik /3.4/ wurde von Beck und Cunningham 1989 auf der Tagung OOPSLA vorgestellt. Die Buchstaben CRC stehen für Class, Responsibilites und Collaborations, sind aber auch die Initialen des Sohnes von Cunningham. Es handelt sich eigentlich um eine sehr einfache Technik, die aber in der Gruppenarbeit sehr erfolgreich ist. Die Grundidee besteht darin, für jede Klasse eine Karteikarte zu nutzen, auf der Verantwortlichkeiten und Beteiligte notiert werden. Die Karteikarte dient dazu, die Informationen

aufzunehmen, die zu einer Klasse gesammelt werden. Sie ist gleichzeitig ein Hilfsmittel, um Objekte einer Klasse zu repräsentieren. Darauf soll aber etwas später noch genauer eingegangen werden. Zunächst werden einige Begriffe geklärt.

Definition 3.8 CRC-Karte

CRC-Karten sind Karteikarten mit dem Namen einer Klasse, ihren Verantwortlichkeiten und ihren Beziehungen.

Definition 3.9 Verantwortlichkeit

Die Menge der interpretierbaren Botschaften zusammen mit den dazu notwendigen Attributen wird als Verantwortlichkeit definiert.

Definition 3.10 Beteiligte

Klassen, die zur Ausführung einer Methode notwendig sind, werden als Beteiligte bezeichnet.

Verantwortlichkeiten sind in diesem Sinne hauptsächlich Methoden, die beschreiben, welche Aufgaben Objekte einer Klasse erfüllen. Die Attribute gehören auch dazu, spielen aber eine sekundäre Rolle. Trotzdem sollten sie mit notiert werden.

Über die Beteiligten wird die Beziehung (Collaboration) zum Restsystem modelliert. Sie repräsentieren Objekte, mit denen eine Kommunikation zur Erfüllung einer gestellten Aufgabe notwendig ist.

Für eine Klasse wird in einer Spalte einer Tabelle notiert, welche Nachrichten an die Klasse gesendet und durch entsprechende Methoden beantwortet werden. In der zweiten Spalte werden zu jeder Methode die Klassen notiert, zu deren Objekten bei der Abarbeitung ein Nachrichtenaustausch notwendig ist.

Im späteren Quelltext entspricht das einem Aufruf einer Methode der entsprechenden Klasse.

Es gibt einige Werkzeuge, die die Anwendung der CRC-Karten-Technik computergestützt ermöglichen. Sie können zum nachträglichen Protokollieren genutzt werden. In der Gruppenarbeit kann ein solches Werkzeug aber die Karteikarte nicht ersetzen. Die folgende **Abbildung 3.6** gibt einen Eindruck, wie eine Karte im System und natürlich auch auf dem Papier strukturiert ist.

Auf der Rückseite der Karteikarte ist die Klasse in Form eines Kommentars definiert. Im System ist das dann ein weiteres Fenster.

Die Gruppenarbeit zur Analyse von Anforderungen mit Hilfe der CRC-Kartentechnik wird wie folgt organisiert. In einer gemeinsamen Sitzung von Fachexperten und Softwareentwicklern wird an alle Beteiligten eine Menge von Karteikarten ausgeteilt. Für bestimmte Anwendungsfälle wird mit Fachexperten über die Notwendigkeit bestimmter Klassen

diskutiert. Immer, wenn eine neue Klasse identifiziert wird, hat derjenige, der den Vorschlag eingebracht hat, eine Karteikarte auszufüllen. Im Verlaufe der gesamten Sitzung bleibt die Verantwortlichkeit bei ihm. Er hat die Aktualisierung am Wissenstand auf der Karte vorzunehmen.

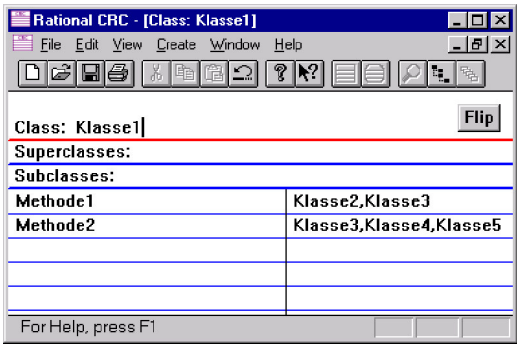


Abbildung 3.6 Aufbau einer Karteikarte

Ist die Diskussion zur Einführung neuer Klassen mit ihren Verantwortlichkeiten zu einem gewissen Zwischenergebnis gelangt, so ist der Zeitpunkt erreicht, um Anwendungsszenarien durchzuspielen.

Simulation der Objektkommunikation

Ein für einen Anwendungsfall typischer Anfang eines Szenarios wird festgelegt. Dabei erfolgt die Auswahl des ersten Objektes, welches der Empfänger der ersten Botschaft ist. Das ist der Anfang eines Sequenzdiagramms, das eventuell bereits spezifiziert wurde.

Das erste Objekt bekommt nun in Form einer Botschaft den Auftrag zur Erfüllung einer Aufgabe. Dazu sind ausreichend Informationen mitzuliefern. Das kann in Form von Parametern der Botschaft, aber auch verbal erfolgen.

Jetzt ist derjenige gefragt, der die Verantwortung für die Klasse des angesprochenen Objektes hat. Er hebt die Karte der Klasse hoch, schlüpft in die Rolle des entsprechenden Objektes und teilt mit, welche Serviceleistungen er von Objekten anderer Klassen benötigt, um auf die an das Objekt gerichtete Botschaft richtig reagieren zu können. Die jeweils verantwortlichen Personen für diese Objekte müssen sich äußern, ob der Service vollständig von dem entsprechenden Objekt erbracht werden kann oder ob es wiederum Serviceleistungen anderer Objekte bedarf. Dabei kann es durchaus vorkommen, dass Anforderungen an Objekte bestehen, für die die gleiche Person verantwortlich ist. Das ist aber unproblematisch. Auf die dargestellte Weise wird das Anwendungsszenario bis zum Ende durchgespielt. Die als notwendig erachteten Methoden werden auf den Karteikarten notiert, falls sie dort noch nicht vorhanden waren.

Bei dieser Art der Interaktion übernehmen Personen die Rollen der Objekte und simulieren das fertige System. Die Kommunikation der Objekte wird durch die Kommunikation zwischen Personen veranschaulicht.