Repository versus Singlesource-Technologie

Frank Sterkmann

Repositories gelangen heute mehr und mehr in die fachliche Diskussion, obwohl sie bereits seit 20 Jahren in der Softwareentwicklung eingesetzt werden. Die Bewertung des Nutzens fällt häufig sehr unterschiedlich aus und ist direkt vom Einsatzzweck abhängig. Neue Technologien, wie die Singlesource-Technologie, drängen auf den Markt und bringen zukunftsorientierte Betrachtungsweisen in den objektorientierten Softwareentwicklungsprozeß. Inwieweit ist die Singlesource-Technologie eine Ablösung für renommierte Repositories, oder ergänzen sich beide Technologien?

Dieser Artikel beschäftigt sich unter anderem mit den folgenden Fragen:

- Was ist ein Repository, und welche Typen von Repositories gibt es?
- Wofür wird ein Repository eingesetzt?
- Wie ist die zukünftigen Entwicklung?
- Wie ist der Nutzen für den Softwareentwicklungsprozeß zu bewerten?
- Welche neuen Technologien und Ansätze für Repositories gibt es?

Modellierungs-Repository

Im umgangssprachlichen Bereich wird der Begriff Repository im Sinne eines strukturierten Datenspeichers verwendet. Bei weiterer Konkretisierung erkennt man die Zuordnung im Softwareentwicklungsprozeß zu den Projektphasen Anforderungsdefinition, Spezifikation und Analyse. Auch im Bereich des Business-Prozeß-Reengineering trifft man vielfach auf den Begriff Repository. Diese Phasen wollen wir als *Modellierungsphasen* und das Repository als *Modellierungs-Repository* bezeichnen.

Dipl.-Ing. MBA Frank Sterkmann ist seit 12 Jahren als Managementberater und Projektleiter sowie seit zwei Jahren als Geschäftsführer bei der Object International Software GmbH in Stuttgart im Bereich der objektorientierten Softwareentwicklung tätig. Seine E-Mail-Adresse lautet: sterkmann@oisoft.com.

Alle diese Anwendungsbereiche haben eines gemeinsam: Sie benötigen ein strukturiertes Medium zur Speicherung abstrakter Anforderungen, Spezifikationen und Analyseergebnisse. Da all diese Informationen eine hohe Vernetzung untereinander aufweisen, vordefinierte Strukturen haben und anhand von Stichwörtern (sogenannten Fachtermini) wiedergefunden werden müssen, werden normalerweise Datenbanksysteme für diese Aufgaben eingesetzt (siehe Abb. 1).

Schnell trifft man bei diesem Thema auf Werkzeuge zur Unterstützung dieser Modellierungsphasen, z. B. CASE-Werkzeuge (Computer-Aided Software-Engineering) oder BPR-Werkzeuge (Business Process Reengineering). Diese Tools gehen über die Verwaltung von textuellen Informationen und Dokumentationen weit hinaus. Sie bieten leistungsfähige graphische Editoren zur Abbildung von statischen und dynamischen Zusammenhängen an und ermöglichen somit eine wesentliche Verbesserung der Übersicht bei abstrakten Softwaresystemen. Die Verwaltung und Bearbeitung von textuellen Informationen zur Beschreibung der Anforderungen erfolgt in Verbindung mit graphischen Modellen zur aussagefähigen Dokumentation des gewünschten Zielsystems. Ferner ermöglichen Konsistenzchecks die einfache Prüfung auf logische Korrektheit des Systems und unterstützen dadurch die Erstellung fachlich vollständiger Anforderungen.

Datenbankgestützte Modellierungs-Repositories weisen eine Reihe von *Vorteilen* auf:

- unternehmensweiter Zugriff auf relevante unternehmensübergreifende Daten
- leistungsfähige Mehrbenutzer- und Zugriffssteuerung
- hohe Datenintegrität und Konsistenz der Informationen
- Abbildung von Vorgehensmodellen und Workflow-Abhängigkeiten

Aber sie bringen auch einige *Nachteile* mit sich:

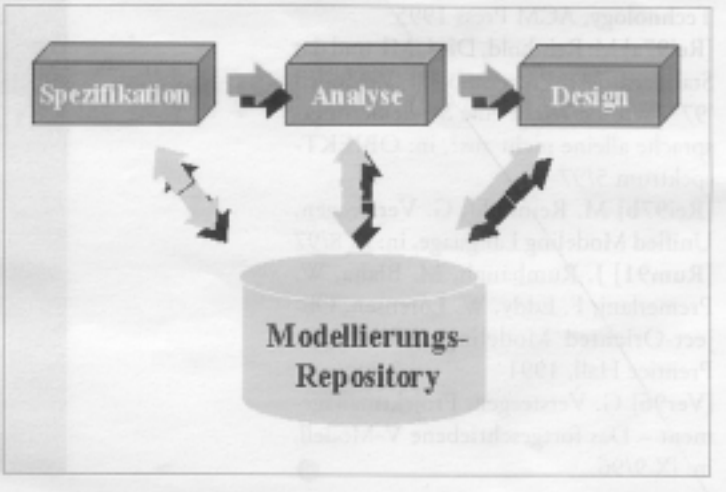


Abb. 1 Modellierungs-Repository

- mangelhafte Revisionsverwaltung von Modulen
- problematische Versionierung über externe Revisionskontrollsysteme
- Synchronisierung von selbständigen, verteilten Entwicklungsgruppen
- Datenaustausch und Synchronisierung mit anderen informationsverarbeitenden Systemen

Quellcode-Repository

Die Modellierungs-Repositories und die zugehörigen Werkzeuge unterstützen primär die frühen Phasen von Softwareprojekten. In den Realisierungsphasen wird bis heute hauptsächlich direkt mit Quellcode auf Dateiebene gearbeitet. Diese Betrachtung gilt für Programmiersprachen der dritten Generation (wie z. B. C++, Java, COBOL) und nicht für 4GL-Entwicklungsumgebungen.

Alle Entwicklungswerkzeuge für 3GL-Sprachen – wie Compiler, Linker, Debugger, Editoren, Make-Systeme und Testwerkzeuge – nutzen Quellcodedateien als Basis. Diese herstellerunabhängige Speicherung von Quellcode hat sich in den letzten Jahren sehr bewährt. Die Verwaltung von Teams und Zugriffen durch mehrere Benutzer wird durch ein unterlagertes Revisionskontrollsystem hervorragend abgedeckt.

Dennoch ist die Steuerung der Granularität der Zugriffe auf die Dateiebene begrenzt. Viele Sprachen haben keinerlei Abhängigkeiten der Dateistrukturen zu den im Quellcode abgebildeten Softwarestrukturen und machen somit eine doppelte Verwaltung von Projektstrukturen notwendig.

Ein neuer Repository-Typ zur Unterstützung der Entwicklungsphasen dringt derzeit auf den Markt. Wir wollen diesen Typ im weiteren Verlauf *Quellcode-Repository* nennen. Einige große Hersteller von Entwicklungsumgebungen nutzen Datenbanksysteme zur strukturierten Verwaltung des Quellcodes in kleinen Einheiten. Innerhalb dieser Entwicklungsumgebungen bringt dies große Vorteile und hohe Steigerungen der Entwicklungsgeschwindigkeit, da eine inkrementelle Kompilierung die sofortige Lauffähigkeit des Zielsystems ohne Wartezeiten ermöglicht (siehe Abb. 2).

Die Nachteile der Quellcode-Repositories sind im Bereich der Ausgrenzung von dateibasierten Entwicklungswerkzeugen zu sehen. Viele leistungsfähige Werkzeuge, wie Make-Systeme, Editoren oder Testwerkzeuge, können nur über den sehr problematischen und umständlichen Weg des Import/Export-Datenaustauschs in den Entwicklungsprozeß eingebunden werden. Ob sich diese Quellcode-Repositories auf dem Markt durchsetzen werden, können wir in den nächsten Jahren beobachten.

Datenbankgestützte Quellcode-Repositories weisen die folgenden *Vorteile* auf:

- fein-granulare Mehrbenutzer- und Zugriffssteuerung auf Klassen, Methoden, Attribute
- hohe Datenintegrität und Konsistenz des Quellcodes
- große Zeitersparnisse während der Entwicklung durch inkrementelle Kompilierung
- Verbesserung der Wiederverwendung durch leistungsfähige Such- und Zugriffsmechanismen

An *Nachteilen* sind zu nennen:

- mangelhafte Revisionsverwaltung von Modulen
- problematische Versionierung über externe Revisionskontrollsysteme

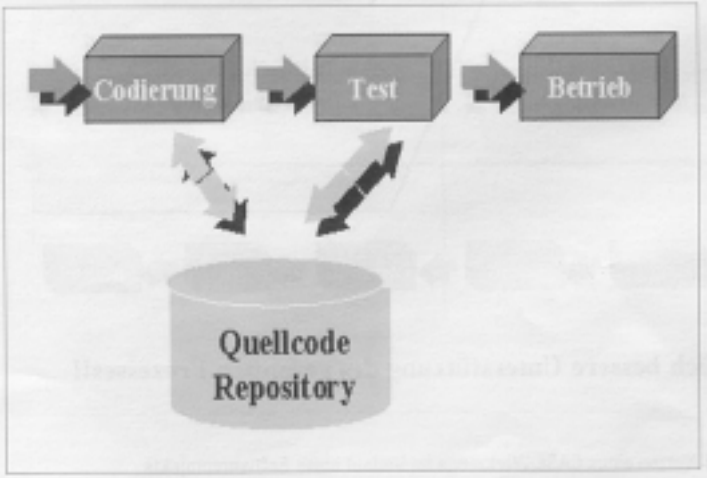


Abb. 2 Unterschiedliche Typen von Repositories zur Unterstützung der Modellierungs- und Realisierungsphasen

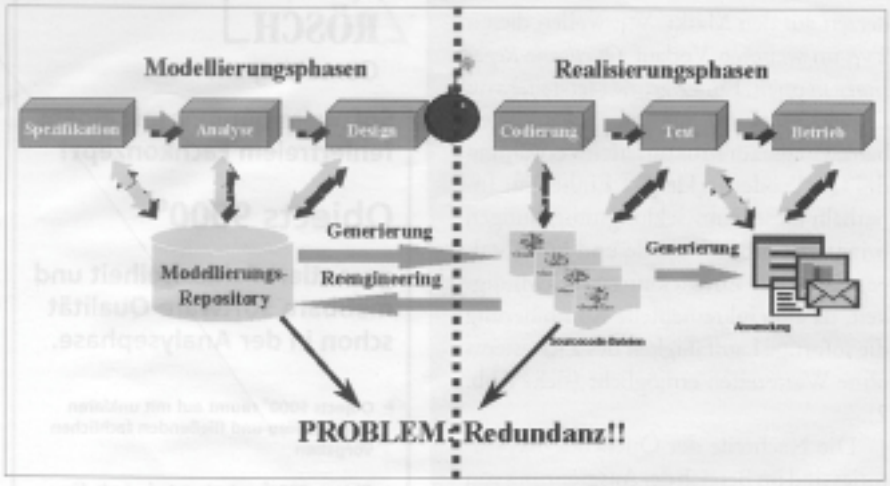


Abb. 3 Redundanz und fehlende Durchgängigkeit im Entwicklungsprozeß

- Ausgrenzung quellcodebasierter Entwicklungswerkzeuge
- Redundanzprobleme beim Im- und Export von Quellcode
- Herstellerabhängigkeit der gesamten Quellcodeverwaltung

Redundanz im Softwareentwicklungsprozeß

Einer der wichtigsten Vorteile der Objektorientierung liegt in der Unterstützung iterativer Entwicklungsprozesse. Die objektorientierte Technologie bietet erstmals die technischen Voraussetzungen für die Durchgängigkeit des Entwicklungsprozesses an, und mit der Objektorientierung werden zum ersten Mal methodische Vorgehensweisen mit einem nahtlosen Übergang von der Analyse über das Design bis zur Implementierung (und vice versa) möglich.

Leider können diese Vorteile in der Praxis nicht genutzt werden, da die heute vorhandenen Entwicklungswerkzeuge immer noch eine strikte Trennung in Modellierungs- und Realisierungsphasen voraussetzen (siehe Abb. 3). Bis auf eine Ausnahme unterstützen CASE-Werkzeuge die Modellierungs-Repositories, und die Entwicklungsumgebungen unterstützen den Quellcode oder eventuelle zukünftig Quellcode-Repositories.

Viele Softwareentwickler sind mit dieser Trennung nicht zufrieden, da eine künstliche Datenredundanz geschaffen wird und der Entwicklungsprozeß in zwei Teile getrennt wird. Um mit der Datenredundanz fertig zu werden, muß erheblicher Aufwand investiert und eine ständige Synchronisierung durchgeführt werden.

Während der ersten Analyse- und Designphase stellt diese noch kein Problem dar, da noch nicht mit quellcodebasierten Werkzeugen – wie Compilern, Debuggern etc. – gearbeitet wird. Für die dann folgende Implementierung wird aber eine Generierung des Quellcodes aus dem Modellierungs-Repository notwendig. Von diesem Augenblick an besteht eine Datenredundanz, die nicht beherrschbar ist, da sich der Quellcode und das Design unabhängig voneinander verändern können.

Auch die Möglichkeit des Reverse-Engineering löst dieses Problem nicht. Häufig wird der Begriff „Roundtrip-Engineering" verwendet, um über diese Grundproblematik hinwegzutäuschen, da Roundtrip-Engineering-Prozesse, wie sie heute angeboten werden, nicht funktionieren. Immer ist ein Zusammenführen (*Merge*) von alten und neuen Strukturen notwendig. Dieser aufwendige Zusammenführungsprozeß ist grundsätzlich nicht automatisierbar, da die semantischen Zusammenhänge von Änderungen nicht automatisch erkannt werden können (siehe Abb. 4).

Diese Problematik dürfte einer der Hauptgründe sein, warum CASE-Werkzeuge zur Unterstützung der frühen Modellierungsphasen bis heute nicht in relevanten Marktanteilen eingesetzt werden. Die ständige Synchronisierung des Quellcodes mit den Modellierungsdaten ist eine Belastung für den Entwicklungsprozeß und wird in den meisten Softwareprojekten nicht konsequent durchgeführt. Da beim Abschluß der Projekte die Anforderungen und Analyse/Design-Daten nicht mehr dem fertigen System entsprechen, wird der Nutzen von CASE-Werkzeugen vielfach in Frage gestellt.

Singlesource-Technologie

Ein neuer Ansatz mit den Namen *Singlesource-Technologie* bietet die grundsätzliche Lösung dieser Problematik an. Die Trennung der Modellierungs- und Realisierungsphasen wird durch die gemeinsame Speicherung aller Informationen des gesamten Entwicklungsprozesses in einem Speichermedium aufgehoben. Mit der einfachen Zusammenlegung der Modellierungs- und Realisierungsinformationen ist es jedoch nicht getan. Wichtig ist die Repräsentation beider Informationen in einem gemeinsamen Metamodell.

Gehen wir etwas in die Tiefe und beginnen beim Ende des Softwareentwicklungsprozesses – der fertig lauffähigen An-

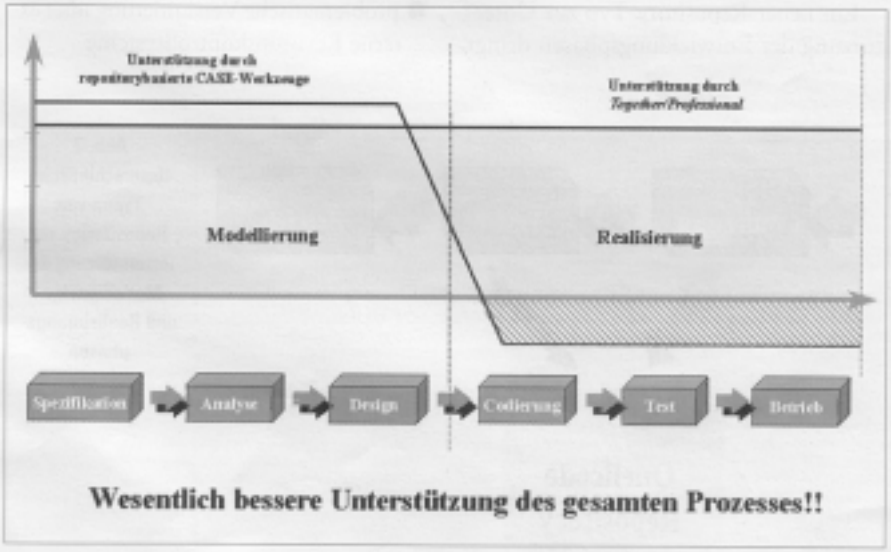


Abb. 4 Nutzen eines CASE-Werkzeugs im Verlauf eines Softwareprojekts

wendung. Die lauffähige Anwendung wird durch einen eindeutig nachvollziehbaren Prozeß aus dem Quellcode erstellt. Der Quellcode stellt damit die einzige Ausgangsbasis dar. Somit muß der semantische Sprachumfang des Quellcodes alle in der lauffähigen Anwendung gewünschten Funktionalitäten abbilden können.

Forward-Engineering mit Singlesource-Technologie

Wenn wir uns in der Argumentation noch einen weiteren Schritt in Richtung Anfang des Softwareentwicklungsprozesses bewegen, muß auch die Aussage richtig sein, daß alle in einer graphischen Modellierungssprache (z. B. der „Unified Modeling Language", UML) nutzbaren Modellierungselemente direkt oder über mehrere Zwischenstufen auf den semantischen Sprachumfang des Quellcodes abbildbar sein müssen. Anders ausgedrückt, muß der semantischen Sprachumfang des Quellcodes eine klare Obermenge der graphischen Modellierungssprache sein. Wäre diese Voraussetzung nicht immer erfüllt, so könnten Anforderungen modelliert werden, die später nicht in eine lauffähige Anwendung abgebildet werden können, was per Definition nicht sehr sinnvoll wäre.

Wenn die oben gemachten Aussagen richtig sind, steht also einer Abbildung – d. h. Speicherung oder Verwaltung einer graphischen Modellierungssprache im Quellcode (C++, Java, Delphi, OO-COBOL etc.) – nichts im Wege (siehe Abb. 5).

Reverse-Engineering mit Singlesource-Technologie

Um jedoch auch beim Reverse-Engineering des Quellcodes die Modellierungselemente korrekt darstellen zu können, muß die Abbildung auch in der Rückwärtsrichtung eindeutig sein. Um vom graphischen Modell zum Quellcode und vom Quellcode zum graphischen Modell zu kommen, müssen daher ein-eindeutige Abbildungsregeln vorhanden sein.

Sinnvollerweise fordern Methodenexperten wie Peter Coad und Jim Rumbaugh die Definition von Regeln, die die eindeutige Abbildung in beide Richtungen definieren, um den Roundtrip-Engineering-Prozeß vollständig unterstützen zu können. Im folgenden werden wir diese Regeln *Blueprints* nennen. Blueprints sind eine Art Template, die zu jedem Notationselement die zugehörigen Quellcodekonstrukte definieren (siehe Abb. 6).

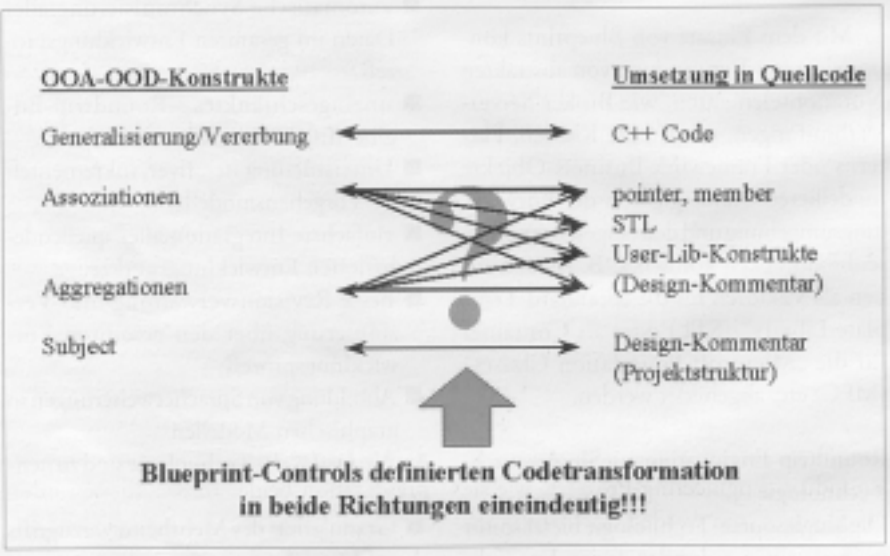
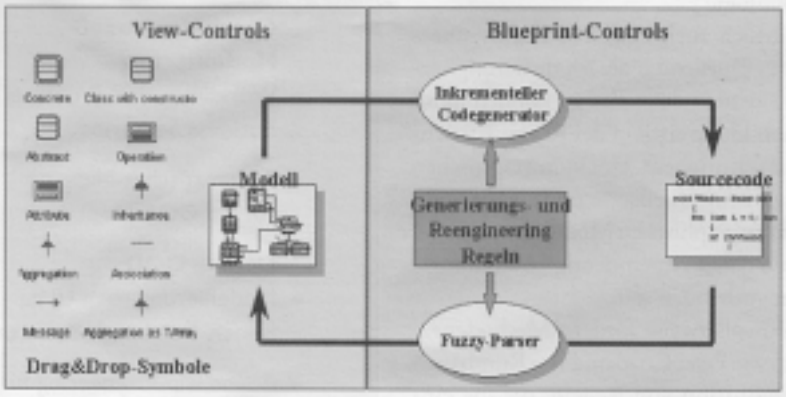


Abb. 5 Ein-eindeutige Zuordnung von Quellcodekonstrukten zu den Notationselementen

Wie Abbildung 5 zeigt, bieten Programmiersprachen meistens mehrere Möglichkeiten zur Umsetzung von Assoziationen oder Aggregationen. Wenn nun für jedes erlaubte Quellcode-Konstrukt ein Blueprint definiert wird, dann können alle Quellcodekonstrukte im Modell dargestellt werden. Möchte man diese Konstrukte auch bei der Modellierung (Forward-Engineering) aktiv einsetzen, so kann man jedem Blueprint ein Darstellungsattribut, wie z. B. Rot, Grün, Blau, Gestrichelt, zuordnen. So können Blueprints bestehende Notationen um quellcode-spezifische Informationen erweitern.



Konfigurierbare Controls als Basis für visuelle Programmierung

Abb. 6 Blueprints zur ein-eindeutigen Abbildung von Quellcode auf graphische Modelle (und vice versa)

Als angenehmer Nebeneffekt kann beim Reengineering auf Wunsch wesentlich mehr Information als bisher gewonnen und im Modell dargestellt werden. Die Steuerung erfolgt frei konfigurierbar durch den Einsatz von Blueprints.

Spracherweiterung durch Bibliotheken, Frameworks und Patterns

Dieses Konzept der Blueprints kann sogar um abstrakte Metastrukturen erweitert werden. Die heutigen 3GL-Programmiersprachen bekommen ihre Attraktivität durch Spracherweiterungen, wie z. B. Klassenbibliotheken, objektorientierte Datenbanken, Broker, GUI-Bibliotheken und Frameworks.

Mit dem Einsatz von Blueprints können sie unter Verwendung von abstrakten Notationselementen, wie Broker-Server-Verbindungen, persistente Klassen, Patterns oder Framework-Business-Objekte modellieren. Abhängig von der Entwicklungsumgebung und den eingesetzten Klassenbibliotheken können z. B. Assoziationen als Vektoren für die „Standard Template Library" (STL) oder als Container für die „Microsoft Foundation Classes" (MFC) etc. abgebildet werden.

Roundtrip-Engineering mit Singlesource-Technologie

Die Singlesource-Technologie bietet somit erstmals einen redundanzfreien Entwicklungsprozeß, beginnend bei der Spezifikation und Analyse, über das Design bis hin zur Implementierung und zum Test des Zielsystems. Es wird eine permanente Synchronisierung und automatische Integrität aller Entwicklungsphasen erreicht, wodurch uneingeschränktes Roundtrip-Engineering erstmals möglich wird (siehe Abb. 7).

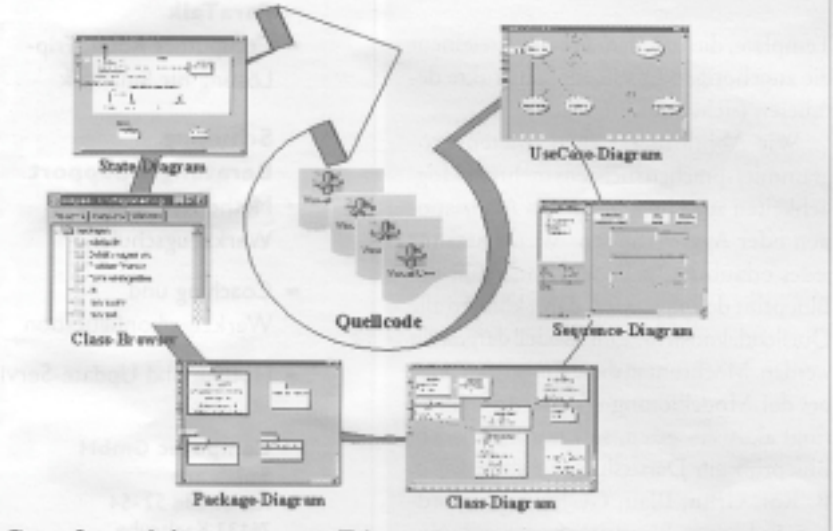
Die Singlesource-Technologie wird derzeit nur von dem Modellierungswerkzeug „Together/Professional" unterstützt. Wie der Name „Together" bereits ausdrückt, werden in diesem Werkzeug die Modellierungs- und Realisierungsphasen vereinigt. Alle graphischen Modelle werden als Sichten auf den Quellcode abgebildet. Somit beherrscht dieses Werkzeug Roundtrip-Engineering par excellence.

Die Singlesource-Technologie hat die folgenden *Vorteile*:

- automatische Synchronisierung aller Daten im gesamten Entwicklungsprozeß
- uneingeschränktes Roundtrip-Engineering
- Unterstützung iterativer, inkrementeller Vorgehensmodelle
- einfachste Integration aller quellcodebasierten Entwicklungswerkzeuge
- beste Revisionsverwaltung und Versionierung über den gesamten Entwicklungsprozeß
- Abbildung von Spracherweiterungen in graphischen Modellen

An *Nachteilen* der Technologie sind zu nennen:

- Granularität des Mehrbenutzerzugriffs auf Dateiebene
- Entscheidung für die Programmiersprache bereits in der Analysephase notwendig



Synchronisierung von Diagrammen und Quellcode

Abb. 7 Unterschiedliche Sichten auf den Quellcode

Resümee

Singlesource-Technologie und Repositories sind keine konträren Ansätze. Ganz im Gegenteil – die Singlesource-Technologie versucht, verschiedene Speicher- oder Abbildungsmechanismen unter einen Hut zu bringen. Wenn wir es schaffen, die Repräsentationen von abstrakten Analyse- und Designinformationen sowie die Strukturen des Quellcodes zusammen zu speichern, um die notwendigen referenziellen Verbindungen zu unterstützen, dann wird der objektorientierte Entwicklungsprozeß an Qualität und Effizienz gewinnen – und dies unabhängig vom Speichermedium Datei oder Datenbank.