Kobold

Produktlinien Management System

Version 1.0

Spezifikation I

Versionsgeschichte

- Version 1.0 (20.02.2004)

  Diese Version wurde dem Auftraggeber vorgelegt.

Inhaltsverzeichnis

1 Einleitung

1.1 Über dieses Dokument

Diese Spezifikation dient als Grundlage zur informellen Beschreibung des Produktlinien Management Systems *Kobold*. Durch das evolutionäre Vorgehensmodell wird dieses Dokument iterativ erweitert, verfeinert und zu Beginn jeder Folge-Iteration der weiteren Entwicklung zugrundegelegt.

Das Dokument richtet sich sowohl an den Auftraggeber und dessen technische Berater als auch an die Mitarbeiter des Werkbold-Teams.

Es wird vom Auftraggeber am Anfang jeder Iteration abgenommen und ist von da an Vertragsbestandteil für die weitere Entwicklung von *Kobold*.

1.1.1 Spezifikation I

Die Spezifikation I dient als Grundlage zur informellen Beschreibung des Rahmensystems von *Kobold*, das in der ersten Iteration entwickelt wird.

Sie basiert auf der Abstraktion der ermittelten Anforderungen aus dem internen Anforderungsdokument für *Kobold* [1], das ständig angepasst wird.

Aufgrund des abstrakten Charakters der Spezifikation des Rahmensystems wird auf eine explizite Spezifikation von Use-Cases verzichtet. Diese wird Gegenstand der Spezifikationen in den Folge-Iterationen sein.

1.2 Das Kobold-System

Das wesentliche Einsatzziel des Produktlinien Management Systems *Kobold* ist die werkzeugunterstützte Entwicklung und Pflege von Software-Produktlinien und die Etablierung eines rollenbasierten Entwicklungsprozesses.

[1] ftp://ftp.berlios.de/pub/kobold/docs/anforderungen/anforderungen.pdf

1.2.1 Grundlegende Architekturentscheidungen

Die grundlegende Architektur von *Kobold* untergliedert sich in zwei Teilsysteme: der Kobold Client, der als rollenbasierte Entwicklungsumgebung für die Verwaltung von Produktlinien und Produkten konzipiert ist, und der Kobold Serverdienst, der die Benutzer-, Rollen- und Nachrichten-Verwaltung ermöglicht.

Der Eclipse-basierte Client bietet dem Benutzer eine graphische Benutzungsoberfläche, über die er seine Produktlinien und Produkte rollenabhängig verwalten kann. Er kann damit seine Architekturen als Graphen ansehen und verändern, neue Rollen verteilen, Nachrichten verschicken und Workflows auslösen.

Der Server verwaltet die Daten der einzelnen Benutzer und deren Zugriffsrechte. Er bietet den Clients außerdem einen zentralen Nachrichtendienst an. Wenn von einem Client eine Nachricht an den Serverdienst gesendet wird, prüft der Serverdienst die Nachricht auf mögliche Konsequenzen, die anderen Clients in Form von Workflows zugeteilt werden. Darüber hinaus verwaltet der Serverdienst auch die Pfade und Zugriffskonfigurationen der Repositories, in denen die Daten der Produkte und Produktlinien gespeichert und versioniert werden.

2 Funktionale Anforderungen der Iteration I

In diesem und den folgenden Kapiteln werden die funktionalen Anforderungen der jeweiligen Iteration beschrieben. Grundlage der Spezifikation der funktionalen Anforderungen im Rahmen der Spezifikation I ist eine sehr abstrakte Beschreibung des *Kobold* Rahmensystems.

In den Folgeiterationen werden zum Kobold Client und Kobold Serverdienst jeweils Use-Cases entwickelt, die die konkret ermittelten und vor jeder Folge-Iteration priorisierten Anforderungen (siehe Anforderungsdokument[1]) detailliert spezifizieren.

2.1 Anforderungen an das Gesamtsystem

Zur rollenabhängigen Verwaltung von Produkten und Produktlinien und zur Etablierung eines Produktlinien-Entwicklungsprozesses ist eine zentrale Schnittstelle zur Produktlinien-, Produkt- und Benutzerspeicherung notwendig, die Rechte, Rollen, Verpflichtungen und Nachrichten der Benutzer bereitstellt und rechnerübergreifend verfügbar ist.

Diese zentrale Schnittstelle in Form des Kobold Serverdienstes wird HTTPS basiert über Remote Procedure Calls von den Kobold Clients über Anfrage- und Antwortmechanismen (Request-Response) angesprochen.

Das *Kobold* Gesamtsystem in Form von mehreren Eclipse basierten Kobold-Client Instanzen und einem zentralen Kobold-Serverdienst hängt von der schnittstellengetreuen Zusammenarbeit der Clients mit dem Serverdienst, sowie der Netzwerkinfrastruktur ab.

Gegenstand der Spezifikation I ist somit, die Grundlage für solide Teilkomponenten zu spezifizieren, die diesen Anforderungen an das Gesamtsystem gerecht werden.

[1] ftp://ftp.berlios.de/pub/kobold/docs/anforderungen/anforderungen.pdf

2.1.1 Rollenprinzip

Zur Etablierung eines Produktlinien-Entwicklungsprozesses werden folgende Rollen von *Kobold* unterstützt:

- Produktlinieningenieur (PLE)
- Produktingenieur (PE)
- Programmierer (P)

Die einzelnen Rollen sind im Begriffslexikon näher erläutert.

Einem Benutzer des Kobold-Systems können mehrere dieser Rollen zugeteilt werden. Die Benutzer- und Rollenverwaltung wird durch den Kobold-Serverdienst gespeichert. Informationen zur genauen Persistierung können aus dem Kapitel zum Kobold-Serverdienst entnommen werden.

2.2 Kobold Client

Der Client basiert grundlegend auf der Eclipse Plattform und deren Standard Widgettoolkit (SWT) und ist dadurch von dessen nativer Schnittstelle abhängig. Er wird als 'Feature-Set' für die Eclipse-Entwicklungsumgebung implementiert und mit einem eigenem 'Product Branding' versehen. Das 'Product Branding' umfasst eine eigene Eclipse-Perspektive mit eigenen Fensternamen und Produktsymbolen, sowie einer Willkommensseite, die einen kurzen Überblick über Funktionalität und Zweck von *Kobold* gibt.

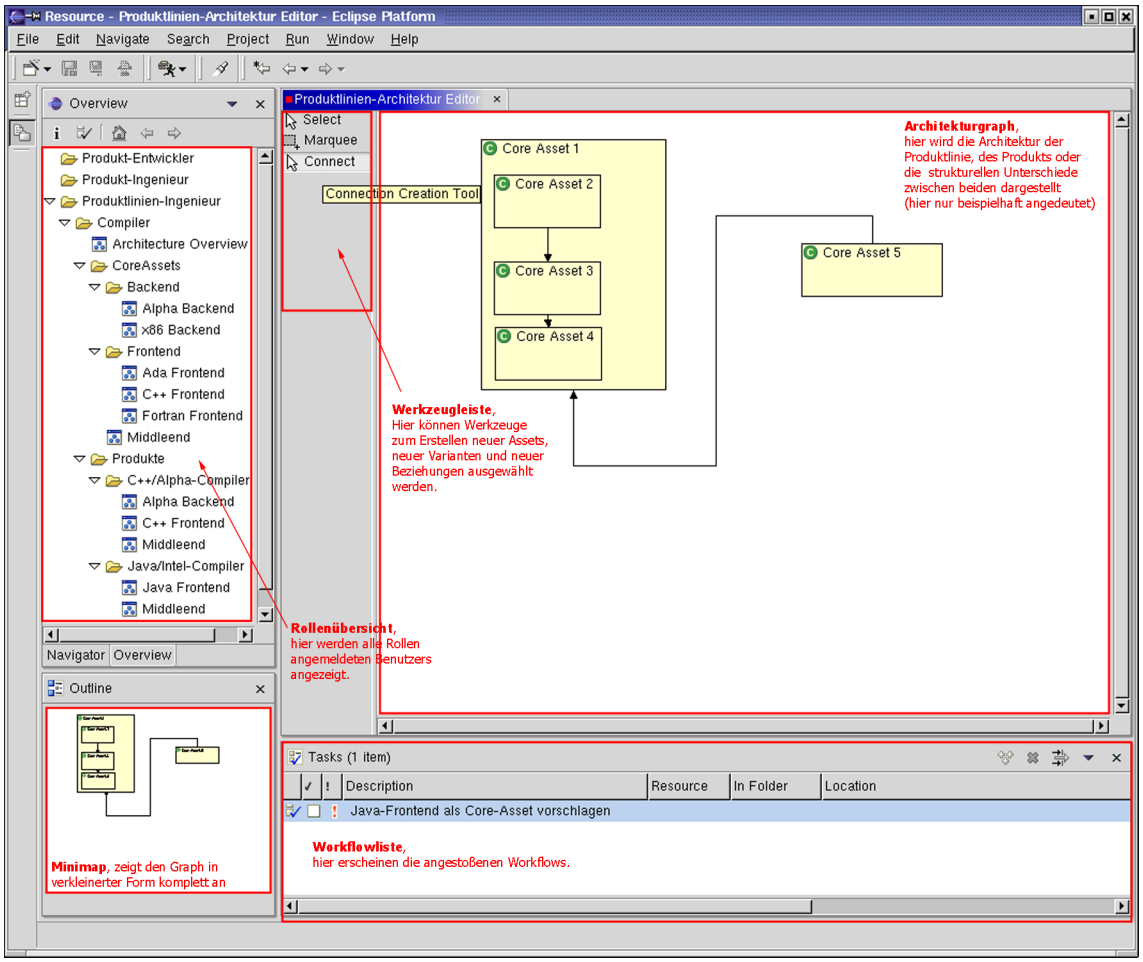

Abbildung 2.1: Screenshot des Client-Prototypen

Das 'Feature-Set' wird als Menge von internationalisierbaren Eclipse Plugins implementiert. Die Ausgangsperspektive dieses Sets besteht aus folgenden Komponenten:

- Der Architektureditor
- Rollen View
- Worklflow/Task View

- Die Minimap

2.2.1 Authentifizierung

Um mit der Kobold-Perspektive in Eclipse zu arbeiten, ist eine zentrale Authentifizierung am Kobold-Serverdienst nötig. Diese wird RPC-basiert über eine SSL verschlüsselte Verbindung realisiert.

Der Benutzer kann bei der Erstbenutzung des Kobold-Clients wählen, ob er sich bei jedem Programmstart authentifizieren will oder ob das Passwort und der Benutzername gespeichert werden sollen (diese Einstellung kann aber auch jederzeit in den Eclipse-Voreinstellungen zur Kobold-Perspektive geändert werden).

Prinzipiell ist der Zugriff auf Produktlinien- bzw. Produkt-spezifische Daten im unauthorisierten Zustand nicht möglich. Auf eine erfolgreiche Authentifizierung reagiert der Client mit der Synchronisation der Produktlinien- bzw. Produkt-spezifischen Repositories und dem Bereitstellen der daraufhin aktualisierten Views für den Benutzer.

2.2.2 Der Architektureditor

In diesem View wird die je nach aktiver Rolle relevante Ansicht auf die Architektur der Produktlinie angezeigt. Diese Architekturansicht wird durch die GEF-basierte Visualisierungsschicht des Architektureditors bereitgestellt, welche komplexe Darstellungs- und Interaktionsalgorithmen zur Visualisierung von Graphen anbietet.

Der Editor bietet die Möglichkeit, grundlegende Operationen durchzuführen, die für die Verwaltung einer Produktlinienarchitektur nötig sind.

Visualisierung von Architekturen

Architekturen werden als Graphen dargestellt, die eine Komposition von Knoten und gerichteten Kanten sind. Knoten enthalten folgende Architekturelemente:

- Komponenten
- Varianten
- Dateien

Die Darstellung dieser Architekturelemente orientiert sich an der in [**?**] beschriebenen Visualisierung und steht endgültig am Ende der Iteration I fest. Rollenabhängig wird es auf der Architekturelement-Ebene eine farblich hervorgehobene Unterscheidung von Elementen geben, für die der aktuelle Benutzer nur Leserechte besitzt.

Gerichtete Kanten stellen Beziehungen mit beliebiger Bedeutung zwischen Architekturelementen dar. Um komplexere Beziehungen modellieren zu können, werden sog. Metaknoten eingeführt, die mehrere eingehende und ausgehende Kanten enthalten können und logische UND- bzw. ODER-Kompositionen von Beziehungen erlauben.

Kanten werden in Form von Linien aus geraden Segmenten mit einem Pfeil dargestellt und können nach Beziehungstyp beliebig eingefärbt werden. Das genaue Aussehen wird durch die Implementierung am Ende der Iteration I festgelegt, orientiert sich jedoch an [**?**].

Metaknoten werden als dreieckige Symbole dargestellt und mit UND- oder ODER-Symbolen beschriftet .Der Knotentyp kann farblich hervorgehoben werden.

Der Architektureditor bietet die Möglichkeit, verschiedene Zoomstufen einzustellen, auch wenn dadurch nur noch ein Ausschnitt der Gesamtarchitektur sichtbar ist.

Um den Überblick über komplexe Software-Architekturen zu erleichtern, stellt eine Minimap die verkleinerte Ansicht der Gesamtarchitektur zur Verfügung, über die auch in der Graphendarstellung navigiert werden kann.

Verwaltung von Architekturen

Architekturen von Produktlinien oder Produkten können durch die Interaktion mit der Maus über Kontextmenüs bzw. Symbolleisten verändert werden.

Zu **Veränderungen** von Architekturen zählen:

- Drag&Drop von Architekturelementen und Beziehungen
- Ändern von Eigenschaften der Architekturelemente oder Beziehungen über einen Einstellungsdialog
- Hinzufügen/Entfernen von Architekturelementen und Beziehungen
- Filtern von Architekturelementen und Beziehungen
- Aufklappen von Architekturelementen
- Detailansicht von Architekturelementen

Darüber hinaus können Unterschiede und Gemeinsamkeiten zwischen zwei oder mehr geöffneten Architektureditor durch einen Algorithmus zur Schnittgraphgenerierung sichtbar gemacht werden. Dabei wird der generierte Schnittgraph in einem neuen Editorfenster dargestellt.

Es gibt die Möglichkeit, über ein Kontextmenü oder eine Tastenkombination die Eigenschaften des ausgewählten Knotens anzuzeigen. Zu diesen Eigenschaften gehören die Metainformationen des Knotens wie z.B. die Liste der schreibberechtigten Benutzer. Näheres zu Metainformationen ist im Abschnitt zur Metainformationskomponente beschrieben.

2.2.3 Die Minimap

Am linken unteren Rand der Ausgangsperspektive wird eine sogenannte Minimap dargestellt, in der ein Überblick über die ganze Architekturansicht der aktuellen Rolle gegeben wird.

Über die Minimap lässt sich im Architektureditor navigieren. Klickt man auf einen Bereich der Minimap, so wird dieser automatisch im Produktlinienarchtikturedi tor zentriert.

2.2.4 Rollen View

Auf der linken Seite der Ausgangsperspektive wird ein hierarchischer Baum angezeigt, der die momentan aktive Rolle und die mit ihr verbundenen möglichen Sichten darstellt. Die Hierarchie ist aus der Darstellung ersichtlich. Rollen werden Produkten bzw. Pro-

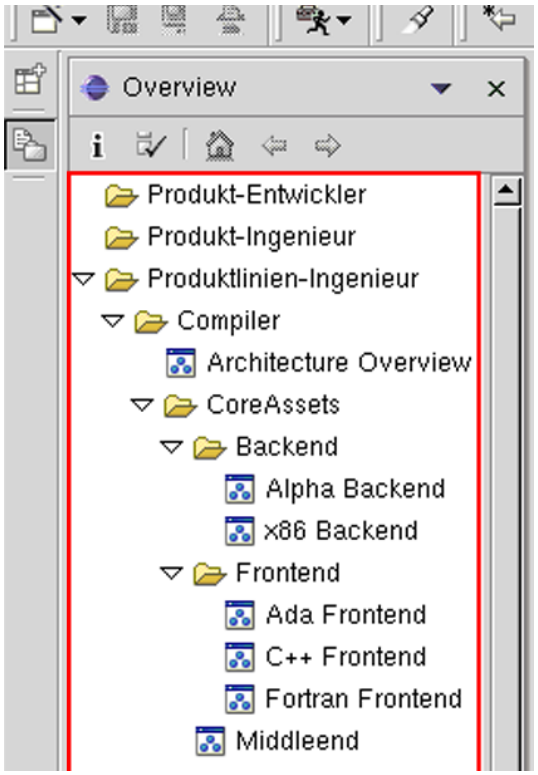

Abbildung 2.2: Prototypische Darstellung der Rollen View

duktlinien zugeordnet, die sich wiederum in Komponenten bzw. Varianten untergliedern, welche sich ihrerseits in Dateien bzw. Verzeichnisse weiter untergliedern lassen. Mit Ausnahme der Rollen der Rollen View und der Querbeziehungen im Architektureditor sind die jeweils dargestelten Architekturen äquivalent.

Ausgehend von den Rollen der Rollen View werden in den folgenden Abschnitten die rollenabhängigen Ansichten auf die Produktlinien- bzw. Produktarchitektur im bereits beschriebenen Architektureditor spezifiziert. Die grafische Notation orientiert sich dabei an der in [**?**] festgelegten Struktur.

Ansicht der Rolle Produktlinieningenieur

Der Produktlinieningenieur hat im Architektureditor standardmäßig eine Ansicht auf die Architektur der gesamten Produktlinie, die sich grafisch sehr stark an die in [**?**] vorgeschlagene Notation anlehnt. Er hat Schreibrechte auf allen Ebenen der Produktlinie und der daraus instanziierten Produkthierarchie.

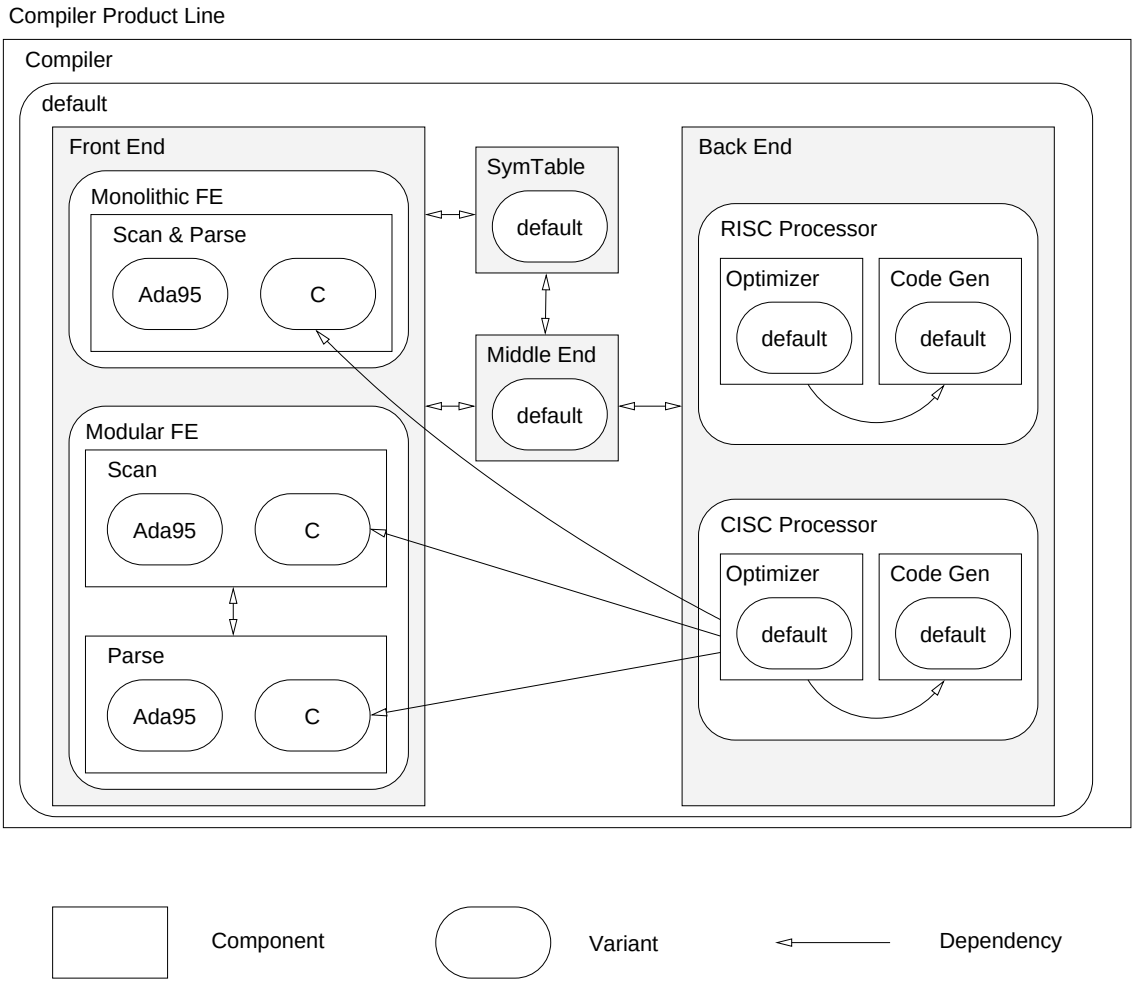

Abbildung 2.3: Ansicht des Produktlinieningenieurs

Ansicht der Rolle eines Produktingenieurs

Der Produktingenieur kann im Architektureditor standardmäßig genau ein Produkt gleichzeitig einsehen. Er kann in der Rollen View die Architekturen der ihm zugeordneten verschiedenen Produkte zur Darstellung auswählen. Graphisch wird sich *Kobold* hier an der graphischen Darstellung aus [**?**] orientieren.

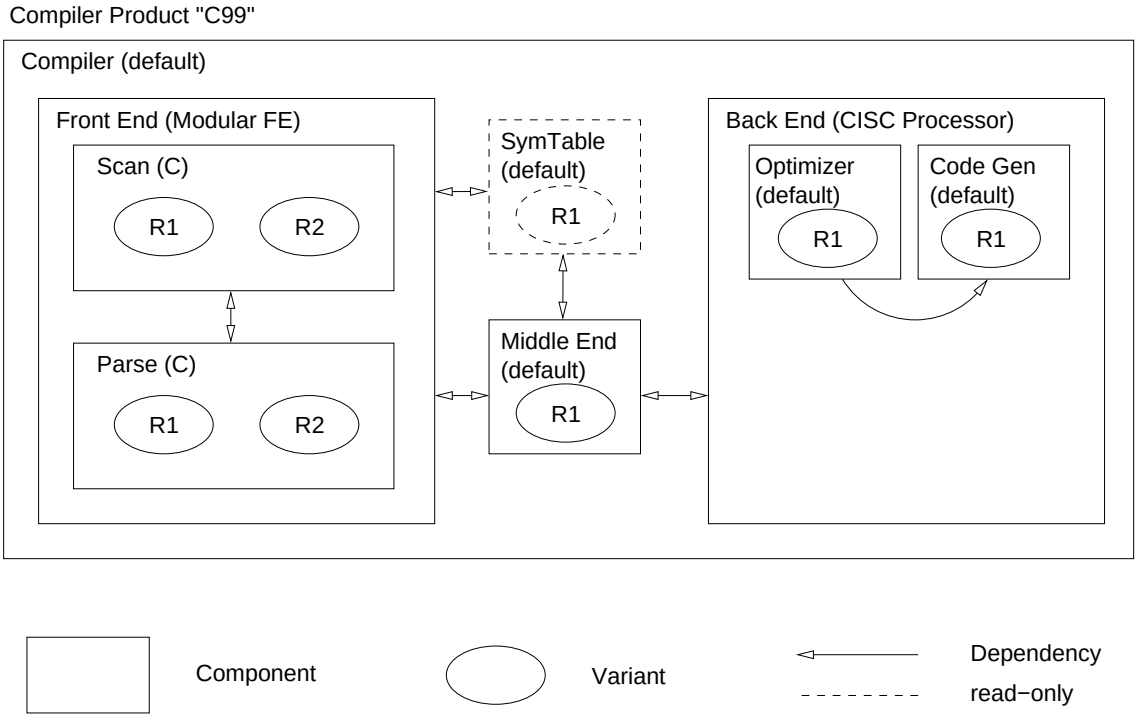

Abbildung 2.4: Ansicht des Produktingenieurs

Ansicht der Rolle eines Programmierers

Der Programmierer hat standardmäßig eine sehr flache Sicht auf ein einziges ihm zugeordnetes Produkt.

Es besteht die Möglichkeit, den aktuellen Stand des Produktes mit seiner aktuellen Arbeitskopie zu synchronisieren.

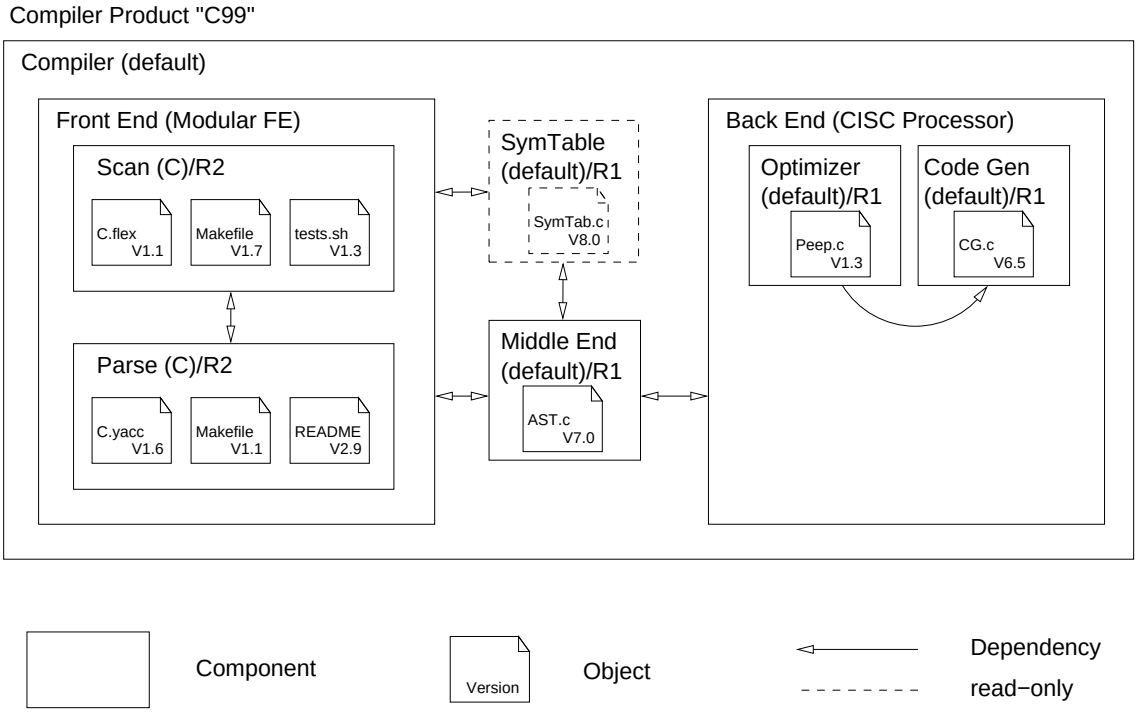

Abbildung 2.5: Ansicht des Programmierers

2.2.5 Worklflow/Task View

Am unteren Rand der Ausgangsperspektive werden standardmässig die noch zu erfüllenden Aufgaben, die aktuellen Nachrichten sowie etwaige Workflows angezeigt.

Durch Doppelklick auf einen Workflow, öffnet sich die jeweilige Detailansicht. Es können Detailansichten zu mehreren Workflows gleichzeitig geöffnet sein.

2.2.6 Workflow Mechanismus

Kobold bietet dem Benutzer einen Workflow-Mechanismus, der ihn bei der Einhaltung des definierten produktlinienbasierten Entwicklungsprozesses unterstützt. Dieser führt Konsistenzprüfungen durch, die in anderen Client-Instanzen automatisch Workflows anstoßen, um erkannte Inkonsistenz zu beheben. Zur Realisierung des Workflow-Mechanismus wird die 'Rule Engine' Drools[2] verwendet, welche eine funktional orientierte, intelligente Wissensbasis modelliert, die auf Fakten Antworten generiert (ähnlich den PROLOG Prinzipien aus der Logik-Vorlesung).

Dafür wird eine Regelbasis in XML für Drools erzeugt. Diese besteht aus einer Menge von Regelmengen und wird vom Serverdienst gespeichert. Eine Regelmenge besteht aus einzelnen Regeln, durch die verschiedene Prüfungen vorgenommen werden können.

Eine solche Regel besteht aus drei Teilen:

Parameter	sind Eingaben (Fakten), die auf die jeweiligen Regeln angewendet werden.
Bedingungen	entscheiden anhand der Parameter, welche Konsequenzen ausgeführt werden.
Konsequenzen	dienen zur Erzeugung/Modifikation von Workflow-Objekten, falls eine Bedingung zutrifft.

Fakten werden auf alle Regeln einer Regelmenge angewendet.

Der Workflow-Mechanismus ist eine Querschnittskomponente zwischen dem Kobold Serverdienst und dem Kobold-Client.

Mechanismus auf dem Kobold Client

Der Kobold Client ruft zu bestimmten Zeitintervallen (Standard: im Minutentakt) Nachrichten vom Server ab. Nachrichten können RPC-Resultate sein, aber auch Workflow-Objekte. Wird ein Workflow-Objekt abgerufen, so erscheint dieses Objekt in der Workflow/Task View und kann per Doppelklick detailliert betrachtet werden.

Der Workflow-Mechanismus kann wie folgt graphisch veranschaulicht werden:

Die Regelbasis in Kobold

Kobold wird nach der ersten Iteration eine Initialregelbasis zur Verfügung stellen, die vom Anwender jederzeit erweitert werden kann. Er muss dabei eine neue Regel erstel-

[2] http://www.drools.org

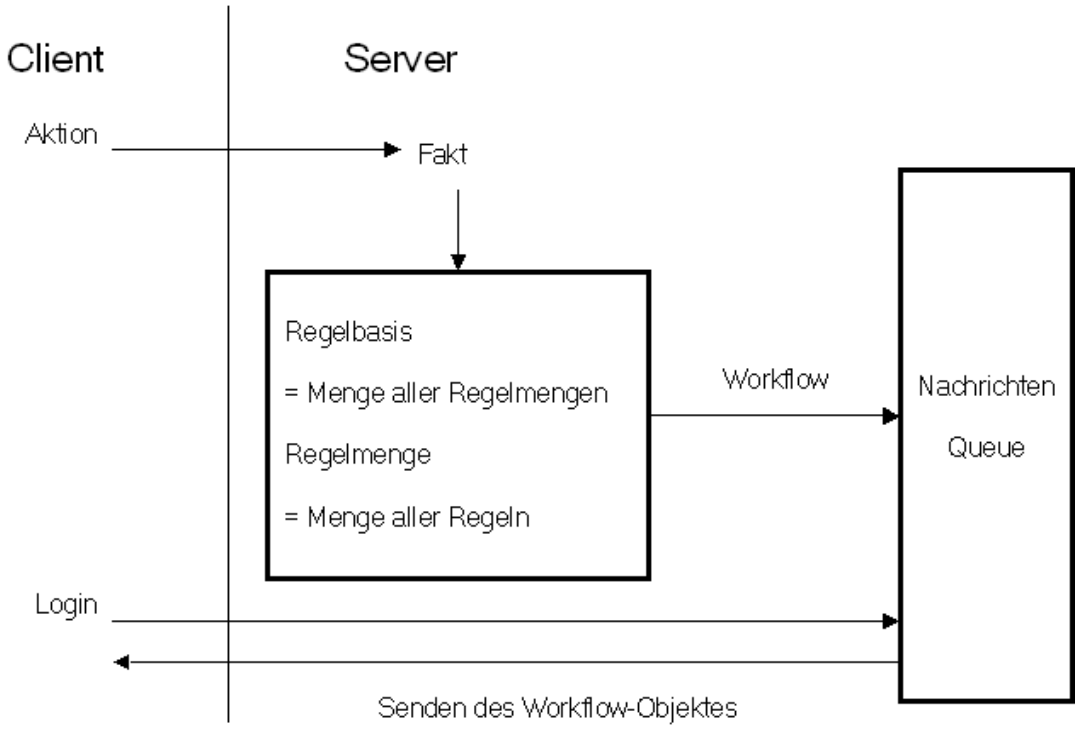

Abbildung 2.6: Workflow Mechanismus

len und sie der Regelbasis hinzufügen. Die Konsequenzen solcher Regeln müssen nicht notwendigerweise Workflow-Objekte erzeugen, sondern können auch andere Aktionen ausführen. Der Mechanismus ist sehr flexibel erweiterbar.

Zur Erstellung einer neuen Regel kann der Workflow-Editor verwendet werden, der in einer späteren Iterationen realisiert wird.

2.2.7 GXL Import und Export Funktionalität

Der GXL Import und Export wird in einem separaten Plug-In realisiert, das zum Kobold-Client 'Feature-Set' gehört.

Es wird sowohl die Metadaten exportieren können, die für die Darstellung der Architekturgraphen benötigt werden, als auch die, die der Darstellung der Dateiebene zugrundliegen.

Diese werden bei einem Export in Form einer XML Datei gespeichert, die dem mit dem Kunden noch zu spezifizierenden GXL Schema entspricht. Die mit der Architektur zusammenhängenden Sourcen werden in einem .jar File exportiert.

2.2.8 Metainformationen

Der Begriff Metainformationen umfasst alle Informationen zu Produkten und Produktlinien, die für das *Kobold* System notwendig sind, jedoch weder aus dem zentralen Kobold Serverdienst noch aus Repository-Informationen gewonnen werden können.

Um die notwendigen Metainformationen der unten aufgeführten Elemente in Kobold zu speichern, wird eine Metainformationskomponente als eigenständiges Plug-In realisiert, das alle Metainformationen verwaltet. Es ermöglicht außerdem eine gezielte Suche nach Elementen mit bestimmten Eigenschaften. Metainformationen werden auch für eine genaue Zuordnung und Identifikation eines Elementes verwendet.

Dieses Kapitel liefert eine genaue Auflistung der geplanten Metainformationen.

Produkt

Für jedes Produkt werden folgende Metainformationen gespeichert:

- Name
- Liste aller Releases

Release

Für jedes Release werden folgende Metainformationen gespeichert:

- Liste der Objekte
- Erstellungsdatum

Version

Für eine Version werden folgende Metainformationen gespeichert:

- interne Versionsnummer
- Status

Objekt

Ein Objekt ist ein beliebiges Softwaredokument (zum Beispiel Quelltext oder Dokumentation), das in der Regel als Datei vorliegt. Für jedes Objekt werden folgende Metainformationen gespeichert:

- Liste der Versionen
- Liste der releasefähigen Versionen
- ID
- Name
- binär (ja/nein)
- Beschreibung
- Zuständiger

Skript

Ein Skript ist ein externer Aufruf einer ausführbaren Datei (z.B. ein Perl Skript, ein configure Skript oder ein externes Programm), welches vor oder nach einer VCM-Aktion ausgeführt wird. Folgende Metainformationen werden dafür gespeichert werden:

- Aktion
- Parameter
- Ausführen vor Aktion (ja/nein)
- Reihenfolge

Variante

Für jede Variante werden folgende Metainformationen gespeichert:

- Liste aller Objekte
- Versionsnummer
- Zuständiger
- Name
- Beschreibung
- ID
- Liste der Skripte
- Status

Komponente

Für jede Komponente werden folgende Metainformationen gespeichert:

- Liste aller Varianten
- Name
- Zuständiger
- Beschreibung
- ID
- Liste der Skripte
- Status

Abhängigkeit

Für jede Abhängigkeit werden folgende Metainformationen gespeichert:

- Typ
- Startknoten
- Zielknoten

Metaknoten

Für jeden Metaknoten werden folgende Metainformationen gespeichert:

- Typ
- ID

Architektur

Für jede Architektur werden folgenden Metainformationen gespeichert:

- Liste der Metaknoten
- Liste der Abhängigkeiten
- Liste der Komponenten (oberste Ebene)
- Name
- Typ
- Status
- Zuständiger
- Link auf das Repository

- Liste der Skripte

VCM Wrapper

Um die Zugriffsrechte auf Repositories des Kobold-Client konsistent durchzusetzen und die automatisierte Nachrichtenzustellung des Kobold Serverdienst zu ermöglichen, wird ein VCM Wrapper erstellt, der das von Eclipse zur Verfügung gestellte Repository Abstraction Layer kapselt.

Der VCM Wrapper fängt VCM-Aktionen an das Repository Abstraction Layer ab und prüft, ob die VCM-Aktionen konform zu den Berechtigungen der aktuellen Rolle des Benutzers sind. Ist dies der Fall, leitet der VCM Wrapper die eigentliche VCM-Aktion an das Eclipse Repository Abstraction Layer weiter. Ist dies nicht der Fall, wird ein Workflow durch den Server angestoßen und die VCM-Aktion verweigert.

Darüber hinaus ermöglicht der VCM-Wrapper die automatische Übernahme von Repository-Zugriffskonfigurationen aus den Produktlinien- oder Produktkonfigurationen, die vom *Kobold* Serverdienst zur Verfügung gestellt werden.

Skriptausführung

Es soll möglich sein, vor und nach jeder VCM-Aktion auf einer beliebigen Architekturebene für alle betroffenen Architekturelemente sog. Skripte auszuführen. Skripte sind externe Aufrufe von ausführbaren Dateien, die mit beliebigen Argumenten versehen werden können. Die Reihenfolge dieser Skripte kann vom Benutzer beliebig bestimmt werden.

Es gibt drei Argumentklassen, die sich wie folgt unterscheiden:

statische Argumente	haben immer einen festen Wert, der dem Skript übergeben wird.
dynamische Argumente	werden als Platzhalter für eine Variable eingesetzt und direkt vor der Ausführung des Skripts von Kobold durch den aktuellen Wert ersetzt. Diese Variablen können z.B. der Modulname, der Name des Verantwortlichen etc. sein. Diese Variablen werden in einer späteren Spezifikation detailiert angegeben.
interaktive Argumente	werden gesammelt, und der Benutzer wird vor der Ausführung des Skripts in einem Dialog nach den Werten für diese Argumente abgefragt. Hier hat er auch die Möglichkeit, die Ausführung des Scripts zu abzubrechen.

2.3 Kobold Serverdienst

Der Kobold Serverdienst koordiniert die Arbeit der Benutzer von *Kobold*. Ein Benutzer ist dabei jede Person mit einem Benutzernamen und dem dazu passenden Passwort. Dieser ist dadurch bei seiner Arbeit nicht an einen festen Rechner gebunden, sondern kann von jedem Rechner mit einem Kobold Client über den Kobold Serverdienst an seine Daten gelangen und diese bearbeiten. Ausserdem ermöglicht der Server eine Client-zu-Client Kommunikation über eine Nachrichten-Queue, welche im Minutentakt von den Clients abgefragt wird.

Der Server wird als HTTPS-basierter *XML-RPC Server* [3] implementiert und bietet damit verschlüsselte Kommunikation. Alle Informationen, die der Server bereitstellt, werden gespeichert.

2.3.1 Berechtigungskonzept

Zur Realisierung des produktlinienübergreifenden Berechtigungskonzepts verwaltet der Server alle benutzer-, rollen-, produktlinien- und produktbasierten Berechtigungen. Dabei kann ein Server produktlinien-übergreifend verwendet werden.

Der Server bietet ein auf diesen Berechtigungen basierenden Authentifizierungsmechanismus an, der von allen Clients verwendet wird. Wird ein Client gestartet, so muss sich der Benutzer über einen Dialog beim Server mit Benutzername und Passwort anmelden. Sind beide korrekt, so werden in seinen Client die Daten der ihm zugeteilten Rollen und den dazu gehörenden Produktlinien bzw. Produkte geladen. Der Benutzer kann dabei nur die Daten verändern, für die er auch eine Berechtigung besitzt.

2.3.2 Persistenz

Die Speicherung aller für das Berechtigungskonzept notwendigen Daten wird durch eine abstrakte, XML-basierte Persistenzschicht, realisiert, wodurch eine flexible Datenhaltung ermöglicht wird.

Die Persistenzschicht dient zur Speicherung aller Rollen-, Authentifizierungs-, Produktlinien- und Produktrelationen. Die Grundstruktur dieser Schicht ist baumartig aufgebaut. Die im Speicher aktuelle Schicht wird nach Ablauf eines konfigurierbaren Zeitintervalls gespeichert (standardmäßig alle 10 Minuten).

In der XML-Struktur werden hierbei die folgenden Elemente gespeichert:

- Benutzer

[3] Nähere Informationen: http://ws.apache.org/xmlrpc/

- Rollen
- Repositories
- Produktlinien
- Produkte
- Relationen

Für jeden Benutzer werden vom Kobold Serverdienst folgende Daten gespeichert:

- Benutzername
- Passwort
- eMail
- Name

2.3.3 Nachrichten-Queue

Der Kobold Serverdienst bildet für jeden Kobold-Benutzer eine Nachrichten-Queue. Werden Nachrichten oder Workflow-Objekte an einen Benutzer geschickt, so werden diese in der Nachrichten-Queue gespeichert und bei der nächsten Nachrichtenabfrage des Benutzers an ihn weitergeleitet.

Die vom Server angebotene Nachrichten-Queue ist zustandsbehaftet, das heißt dass bei nicht physisch begründeten Ausfällen die Nachrichten-Queue in der Regel ohne Datenverlust wiederherstellbar ist.

Für jede Nachricht in der Nachrichten-Queue werden folgende Daten benötigt:

- Absender
- Empfänger
- Inhalt
- Datum
- ID
- Priorität

Workflow-Mechanismus auf dem Kobold Server

Sendet ein Client dem Server eine Nachricht, wendet dieser diese Nachricht als Fakt auf eine kontextabhängige Regelmenge aus der Regelbasis an. Dabei werden alle Regeln der Regelmenge durchlaufen. Gilt eine Bedingung einer Regel, so wird die zugehörige Konsequenz ausgeführt. Im Implementierungsteil einer solchen Konsequenz wer-

den Modifikationen an einem erzeugenden Workflow-Objekt durchgeführt (z.B. wird bei einem unauthorisierten Commit ein String “Unauthorisierter Commit“ in das Workflow-Objekt geschrieben, sofern eine Regel dies definiert hat).

Ein Workflow-Objekt ist eine XML-Struktur, die einen Titel und eine Liste aller Regelresultate besitzt. Dieses Workflow-Objekt wird einem Client zugeteilt und in die Message-Queue des Servers gelegt.

3 Nichtfunktionale Anforderungen

3.1 Leistungsanforderungen

Das System wird zunächst so ausgelegt, dass mind. 50 Benutzer gleichzeitig mit dem Produkt arbeiten können. Die Skalierbarkeit hängt ausschliesslich vom Server ab, dessen Leistungsfähigkeit von der Hardware und der verwendeten Persistenzschicht- Implementierung beeinflusst wird. Garantiert wird die Bereitstellung von 10 Produktlinien zu je 50 Produkten. Pro Produktlinie können 100 Benutzer verwaltet werden.

3.2 Minimale Hardwareanforderungen

Das Softwaresystem wird auf einem Pentium II basierten PC mit 400 Mhz CPU-Takt, 256 MB Hauptspeicher und mind. 200 MB freien Festplattenspeicher (oder unter einer vergleichbaren Unix-Workstation) unter den Betriebssystemen Windows, Solaris und Linux lauffähig sein.

3.3 Entwurfseinschränkungen

Das Produkt wird in Java 2 Version 1.4 implementiert. Als Workbench-Framework wird die Eclipse Plattform[1] verwendet, sowie deren Widgettoolkit SWT[2] und GEF[3]. Zudem werden Bibliotheken von der Apache Software Foundation[4] benutzt. Die Generierung der Dokumenation setzt auf iText[5] auf. Für die SSL-verschlüsselte Kommunikation mit dem Server kommt die JSSE von Sun zum Einsatz. [6]

[1] http://www.eclipse.org/, http://www.eclipse.org/platform/
[2] Standard Widget Toolkit, http://www.eclipse.org/swt/
[3] Graphical Editing Framework, http://www.eclipse.org/gef/
[4] http://www.apache.org/
[5] http://www.lowagie.com/iText/
[6] Java Secure Socket Extension, http://java.sun.com/products/jsse/index.jsp

3.4 Verfügbarkeit

Für den Client gibt es keine besonderen Anforderungen bzgl. der Verfügbarkeit. Der Kobold-Serverdienst soll mit mind. 99% hoch verfügbar sein. Die Server-Verfügbarkeit hängt trotzdem von der Konfiguration des Rechners, der den Serverdienst zur Verfügung stellt, ab.

3.5 Sicherheit

Die vom Server bereitgestellten Daten werden bei jeder Änderung sofort auf dem Datenträger gespeichert. Somit wird die Gefahr eines Datenverlusts beim Eintritt von unvorhersehbaren Ereignissen minimiert. Alle client-spezifischen Daten werden in den Produktlinien- bzw. Produkt-Repositories gespeichert und unterliegen den Sicherheitsregeln des verwendeten Versionskontrollsystems.

3.6 Robustheit

Fehleingaben und Fehler im System werden erkannt und dem Benutzer mit einer aussagekräftigen Fehlermeldung mitgeteilt. Sie führen nicht zum Programmabsturz. Genaueres folgt in einer späteren Iteration.

3.7 Wartbarkeit

Durch die ständige Überprüfung des Programmquellcodes mit Metriken auf Kopplung, Zusammenhalt und Styleguide-Konformität wird die hohe Wartbarkeit des Produkts gewährleistet.

3.8 Performance

Es wurde nicht gefordert den ganzen Graphen, der unter Umständen sehr komplex werden kann, performant zu visualisieren.

Es soll genügen, die für die jeweilige Architekturansicht relevanten Ausschnitte des Graphen zu visualisieren. Diese werden sich der Übersichtlichkeit pro forma in einer Größenordnung bis zu 125 Knoten bewegen. Die Anzeige der Teilgraphen sollte eine Grenze von 8 Sekunden zur Berechnung der Darstellung nicht überschreiten.

3.9 Portabilität

Der Server stützt sich auf keine nativen Schnittstellen sondern benutzt ausschließlich *pure java*. Somit ist er laut Sun Microsystems Inc.[7] unter Solaris-SPARC, Solaris-Intel, Windows NT/2000/XP und Linux lauffähig.

Der Client basiert grundlegend auf der Eclipse Plattform und deren Widgettoolkit und ist dadurch von dessen nativer Schnittstelle abhängig. Laut dem Eclipse Consortium werden die folgenden Plattformen und Betriebssysteme unterstützt:

- Windows NT/2000/XP
- Linux (Motif)
- Linux (GTK 2)
- Solaris 8 (SPARC/Motif)
- QNX (x86/Photon)
- AIX (PPC/Motif)
- HP-UX (HP9000/Motif)
- Mac OSX (Mac/Cocoa)

[7] unter http://java.sun.com/j2se/1.4.2/system-configurations.html

A Begriffslexikon

- **Ansicht** Eine rollenabhängige Sicht
- **Arbeitskopie** Kopie, mit der gearbeitet wird und die tatsächlich verändert wird
- **Architektur** Struktur von Software, die aus Komponenten und Konnektoren besteht
- **Assets** Wiederverwendbare Komponenten, z.B. Source-Pakete, Klassen, usw.
- **Auschecken** VCM-Aktion zum Anfordern einer Arbeitskopie aus dem Repository
- **Core-Assets** Architekturkomponenten, die bestimmte Funktionalitäten erfüllen und oft wiederverwendet werden. Sie bilden mit den produktspezifischen Komponenten das Produkt
- **Core-Asset-Entwickler** Softwareentwickler, der Core-Assets entwickelt
- **Core-Asset-Repository** Entwicklungs-Repository (Arbeitskopie) der Core-Assets
- **Deprecated** Veraltet, missbilligend
- **Einchecken** VCM-Aktion zum Rückschreiben der Arbeitskopie-Änderungen ins Repository
- **Entwicklungs Repository** Siehe Arbeitskopie
- **Erstentwicklung** Entwicklung eines neuen Produkts bzw. eines neuen Core-Assets
- **Evolutionäres Vorgehensmodell** Inkrementelles Vorgehensmodell in der Softwareentwicklung, ähnlich dem Spiralmodell nach B.W. Boehm
- **Fakt** eine Mitteilung an den Kobold-Server
- **Feature-Set** Ein Plugin-Satz zur Erweiterung von Eclipse, so dass Eclipse nach der Erweiterung ein eigenes Erscheinungsbild hat
- **Iteration** Phase im evolutionären Vorgehensmodell
- **Kernkomponenten** siehe Core-Assets
- **Komponenten** Bestandteile der Produktlinien- bzw. Produktarchitektur
- **Message-Queue** Nachrichten, die auf Anfrage an den Benutzer geleitet werden
- **Metainformation** Zusatz-Informationen über Informationen
- **Module** siehe Komponenten

- **Plugin** Softwarekomponente, die die Funktionalität des Systems erweitert
- **Produktarchitektur** Software-Architektur eines Produktes
- **Produkt-Entwickler** P, Softwareentwickler dessen Vorgesetzter ein Produkt-Ingenieur ist
- **Produkt-Ingenieur** PE, ein Software-Ingenieur mit Leitungs- und Entscheidungskompetenz bzgl. der technischen Realisierung des Produkts. Einem Produkt-Ingenieur ist in der Regel ein Produktlinien-Ingenieur vorgesetzt.
- **Produktlinien-Ingenieur** PLE, Software-Ingenieur mit projektübergreifender Leitungs- und Entscheidungskompetenz bzgl. der technischen Realisierung und Pflege der Produktlinienarchitektur.
- **Produkt-Repository** Repository, das vom PE verwaltet wird, Entwickler checken Arbeitskopien von ProduktKomponenten aus
- **Produktlinien-Repository** Repository, das vom PLE verwaltet wird, PE checken Arbeitskopien von Core-Asset Varianten aus
- **Prototyp** Entwicklungsversion eines Programmes
- **RPC** Remote Procedure Call erlaubt die Ausführung von Funktionen, die auf einem anderen Rechner implementiert sind
- **Repository** Datei-Verwaltung zur Verwaltung von Versionen
- **Softwarearchitektur** siehe Architektur
- **Update** VCM-Aktion, die die Arbeitskopie mit dem Repository synchronisiert
- **VCM** Version Control Management, ein Versionsverwaltungssystem wie z.B. das Versionsverwaltungstool CVS (Concurrent Versions System)
- **VCM-Aktion** Version Control Management Aktion, z.B. Auschecken, Einchecken, Synchroniseren, Updaten
- **Varianten** Modifikationen von Core-Assets
- **Version** Eindeutige Bezeichnung zu einem fixen Zeitpunkt
- **View** Subfenster von Kobold-Client
- **Wasserfall-Vorgehensmodell** Klassisches Vorgehensmodell in der Softwareentwicklung, das in Phasen aufgeteilt ist, die aufeinander folgen
- **Workflow** Arbeits- bzw. Geschäftsprozess

Literaturverzeichnis

[1] Daniel Simon, Thomas Eisenbarth, Stefan Bellon, Gunther Vogel, Jörg Czeranski *PLAM - A Product Line Asset Management Tool*