Universität Hamburg

Fachbereich Informatik

Diplomarbeit

Anforderungen an ein EDV-gestütztes Software-Konfigurationsmanagement am Beispiel eines Client/Server Rückversicherungsinformationssystems

Juni 2000

Bertrand Scheller
Lokstedter Weg 112
D-20251 Hamburg

Matr.Nr.: 3 941 672
Tel.: 040-46073933

Betreuung:	Prof. Dr. Klaus Brunnstein, Arbeitsbereich Anwendungen der Informatik in Geistes- und Naturwissenschaften (AGN)
Zweitbetreuung:	Dr. Ralf Klischewski Arbeitsbereich Softwaretechnik (SWT)

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AIDS	**a**cquired **i**mmuno**d**eficiency **s**yndrome
CB	PVCS Configuration Builder
CCB	Change-Control-Board
DGQ	Deutsche Gesellschaft für Qualität e.V.
DDL	data definition language
DLL	dynamic link library (dynamische Link-Bibliothek)
DML	data manipulation language
DV	Datenverarbeitung
EDV	Elektronische Datenverarbeitung
EN	Europäische Norm
EXE	executable (ausführbares Programm)
GmbH	Gesellschaft mit beschränkter Haftung
IBM	International Business Machines Corporation
IDE	Integrated Development Enviroment
ISO	International Organization for Standardization Internationale Standardisierungsorganisation
IT	Informationstechnologie
KE	Konfigurationselement(e)
LAN	Local Area Network
OS/2	Operating System/2 (Betriebssystem der IBM)
PC	Personal Computer
PM	Projektmanagement
PRORIS	Professionelles Rückversicherungs-Informationssystem
QM	Qualitätsmanagement
RV	Rückversicherung
SKM	Software-Konfigurationsmanagement
SQL	Structured Query Language
Tracker	PVCS Tracker
VM	PVCS Version Manager
WAN	Wide Area Network

1 Einleitung

1.1 Motivation für die Diplomarbeit

In den letzten Jahren hat Softwarequaliätssicherung zunehmend an Bedeutung gewonnen. Dies hat unterschiedliche Ursachen. Die Aufgaben von DV-Systemen werden immer komplexer und in ihren Auswirkungen auf den Menschen und seine Umwelt zunehmend krititischer. In fast allen Lebensbereichen werden die Menschen mit DV-Systemen direkt oder indirekt konfrontiert. Es wird häufiger hinterfragt, ob die eingesetzten Systeme vertrauenswürdig sind und wie dieses sichergestellt werden kann. Softwarequalität ist damit über Technikfolgenüberlegungen zum Gegenstand der Diskussion in der Gesellschaft geworden. Aktuelles Beispiel für diese Überlegungen ist das "Jahr 2000 Problem", das auf beeindruckende Art und Weise die Abhängigkeit unser heutigen Gesellschaft von informationsverarbeitenden Systemen und deren Qualität deutlich macht.

Bezogen auf die Softwarequalitätssicherung ist dem Kostenfaktor bei der Softwareentwicklung besondere Beachtung zu schenken. Untersuchungen zeigen, daß 75% der Kosten eines Softwareprojekts der Wartungsphase zugeordnet werden müssen, da der Aufwand für Änderungen an einem bestehenden System vielfach wesentlich höher als für Neuentwicklungen ist [Dunn 93]. Dies hat meist seine Ursache in einem unzureichenden und wenig dokumentierten Softwareentwicklungs- bzw. Änderungsprozeß. Ein wesentliches Instrument zur Verbesserung dieser Prozesse kann ein Software-Konfigurationsmanagementsystem (SKM-System) sein.

Hier bietet sich ein Ansatzpunkt um Qualitätsprobleme über alle Entwicklungsphasen des Projekts hinweg zu vermeiden und besonders in der Wartungsphase ein Softwareprodukt effizient im Sinne von Zeit, Kosten und Qualität pflegen bzw. weiterentwickeln zu können.

Die Einführung und Ausgestaltung von SKM-Systemen wird aber nicht immer von allen Projektbeteiligten positiv gesehen. Häufig wird mit SKM-Systemen ein übertriebener Bürokratismus in Verbindung gebracht, der den Softwareentwicklungsprozeß behindert und die Projektbeteiligten in ihrer kreativen Entwicklungsarbeit einschränkt. Um für SKM-Systeme eine breite Akzeptanz bei allen Projektbeteiligten zu finden, ist es notwendig die unterschiedlichen Anforderungen der Projektbeteiligten an ein SKM-System zu betrachten und aufeinander abzustimmen.

1.2 Problem- und Aufgabenstellung

Organisation und Kontrolle eines Softwareentwicklungsprojekts haben wesentlichen Einfluß auf die Qualität des Softwareentwicklungsprozesses und des entwickelten Produkts. Eine zentrale Rolle im SKM spielt die ständige Überprüfung von Status und Fortschritt der Produktentwicklung im Projektverlauf. Die meistens in Teamarbeit durchgeführten Softwareentwicklungsprojekte sind durch arbeitsteilig organisierte Prozeßabläufe äußerst komplex und bedürfen der Organisation und Kontrolle, um die übergeordneten Entwicklungsziele des Projekts zu erreichen.

In der Wartungsphase ist es besonders wichtig, die einmal erreichte Qualität zu erhalten. Bei umfangreichen Softwaresystemen die aus einer großen Anzahl von Elementen - wie z.B. Dateien, Datenbanken, Moduln, Spezifikationen, Testfällen und Testergebnissen - bestehen, zwischen denen eine große Anzahl von Beziehungen herrschen, ist es ohne die Unterstützung durch Werkzeuge kaum möglich die Elemente konsistent zu verwalten. Dies wird besonders deutlich durch Untersuchungen, die zeigen, daß 53% der Fehler durch Auswirkungen von Änderungen auf bereits existierende Produktteile entstehen [Coll 87]. Dieser Umstand ist ein wesentlicher Grund für hohe Wartungskosten in Softwareprojekten.

Im Rahmen dieser Diplomarbeit werden Anforderungen aus Sicht der unterschiedlichen Projektbeteiligten an ein SKM-System herausgearbeitet, analysiert und potentielle Zielkonflikte identifiziert. Am Aufbau des konkreten SKM-Systems für das Client/Server Rückversicherungs-Informationssystem PRORIS (PRORIS) der Firma CTH Consult TEAM Hamburg GmbH, wird geprüft, inwiefern die zuvor aufgestellten Anforderungen - im Umfeld kommerzieller Anwendungsentwicklung für Finanzdienstleister - erfüllt werden und welche Verbesserungspotentiale vorhanden sind. Des weiteren wird untersucht, inwiefern die Funktionalität und Integrationsfähigkeit der in PRORIS verwendeten SKM-Werkzeuge der PVCS-Familie (PVCS ist ein SKM-Produkt der Firma MERANT), Einfluß auf die zuvor definierten Anforderungen und damit auch auf die Akzeptanz des SKM-Systems hat.

1.3 Aufbau der Diplomarbeit

In dieser Diplomarbeit wird das Thema Software-Konfigurationsmanagement anhand von Literatur und am Beispiel eines konkreten SKM-Systems aufgearbeitet.

Im ersten Kaptitel erfolgt eine Einführung in das Thema Rückversicherung und in das PRORIS-System.

Anschließend wird im zweiten Kapitel ein allgemeiner Überblick über das Thema Software-Konfigurationsmanagement in der heutigen wissenschaftlichen Diskussion gegeben sowie das zu betrachtende Themenfeld für diese Diplomarbeit weiter eingegrenzt. Am Ende des zweiten Kapitels werden Anforderungen an ein SKM-System aus Sicht unterschiedlicher Benutzerrollen im PRORIS-Team erarbeitet.

In den folgenden Kapiteln drei, vier und fünf wird der Aufbau eines konkreten SKM-System für das Client/Server Rückversicherungssystem PRORIS im Spiegel der zuvor aufgestellten Anforderungen betrachtet. Dabei wird untersucht inwiefern die Anforderungen erfüllt werden konnten. Darüber hinaus werden ggf. vorhandene Zielkonflikte bei den Anforderungen herausgearbeitet sowie Einschränkungen, die sich aus der Verwendung der SKM-Werkzeuge ergeben analysiert.

Im Kapitel sechs wird das Ergebnis der Untersuchung zusammengefaßt sowie Verbesserungspotentiale für das SKM-System von PRORIS dargestellt. Zusätzlich wird ein Ausblick auf die sich abzeichnenden Entwicklungen im Bereich des Software-Konfigurationsmanagement gegeben.

1.4 Rückversicherung

Versicherungen bewahren Menschen oder Unternehmen gegen Entgelt vor wirtschaftlichen Nachteilen durch Ereignisse wie Krankheit, Unfall, Feuer, Diebstahl etc.. Der Versicherte ersetzt somit variable Kosten, Kosten im Schadenfall, deren Ausmaß und zeitlicher Anfall im vorhinein nicht bekannt ist, durch fixe Kosten, Prämien, mit denen er kalkulieren kann[1].

Ein Versicherer kann nun seinerseits einen Teil der von ihm übernommenen Risiken auf einen Rückversicherer abwälzen. Man kann also eine Rückversicherung (RV) als „Versicherung des Versicherers“ bezeichnen. Die folgende Abbildung zeigt anschaulich, wie die internationale Rückversicherung und Weiter-Rückversicherung (Retrozession) aussehen kann, so daß auch große Risiken für die einzelnen Beteiligten tragbar werden.

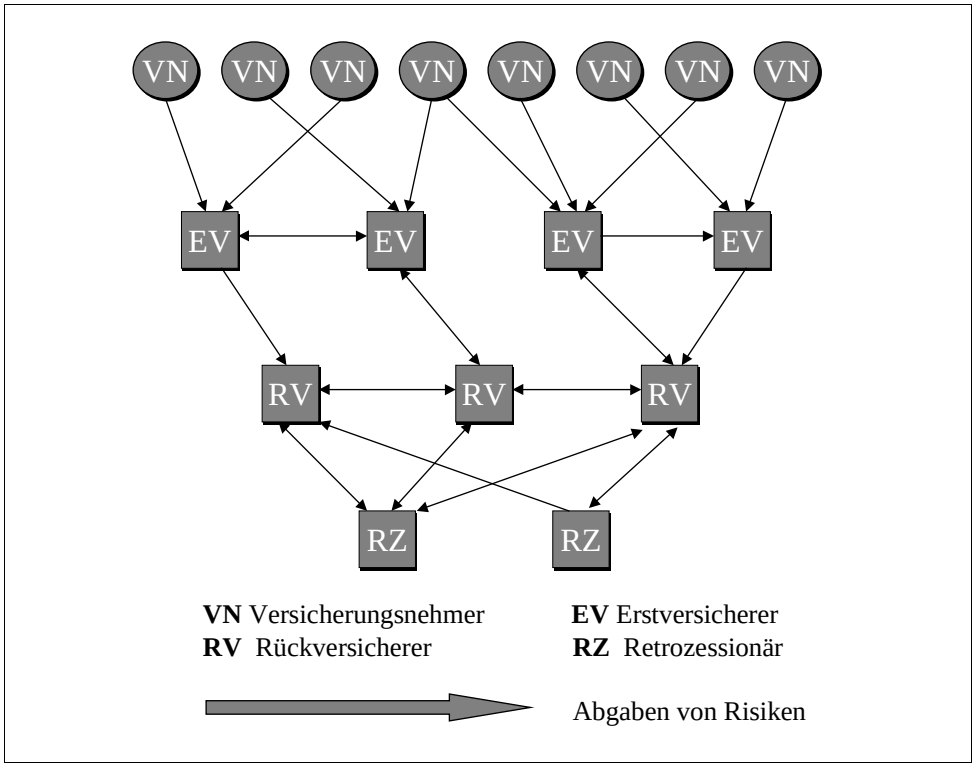


Abb. 1-1: Schema der Risiko-Aufteilung[2]

[1] vgl. Gerathewohl, Klaus u.a., Rückversicherung Band 1, S. 1
[2] Grossmann, Marcel, Rückversicherung, S. 8

Der Rückversicherungsmarkt

Der gesamte Versicherungsmarkt kann definiert werden als „Gesamtheit der Beziehungen zwischen den Beteiligten aufgrund freier Angebots- und Nachfragedisposition der Dienstleistung Versicherung“[3].

Der Rückversicherungsmarkt ist ein Teilbereich des Versicherungsmarktes. Er ist von seiner Natur her international ausgerichtet, da Rückversicherer gerade wegen des Größtschadenrisikos einen geographischen Ausgleich seines Versicherungsbestandes benötigt.

Nachfrage nach Rückversicherungsschutz besteht sowohl bei Erstversicherungs- wie auch bei Rückversicherungsunternehmen, die ihrerseits rückversicherte Risiken rückversichern wollen.

Anbieter von Rückversicherungsschutz sind gemischte Versicherungsunternehmen die Erst- und Rückversicherung betreiben. Weitere Anbieter sind professionelle Rückversicherungsunternehmen, die ausschließlich indirektes Geschäft, d.h. Rückversicherung, betreiben. Zu den traditionellen Anbietern und Nachfragern haben sich seit einiger Zeit sogenannte Captive Companies gesellt, die meist hundertprozentige Töchter von Konzernen sind und ursprünglich die Risiken ihrer multinationalen Muttergesellschaften decken sollten, aber teilweise auch fremdes Geschäft zeichnen. Bei Lloyd’s in London wird Rückversicherung sogar in der Börse betrieben

Schließlich spielen die RV-Makler auf dem Markt eine Rolle, die RV-Geschäft zwischen Erstversicherern und Rückversicherern vermitteln. Die Quantifizierung des Marktvolumens erweist sich als problematisch, da neben den professionellen Rückversicherungsunternehmen auch die gemischten berücksichtigt werden müssen und in einigen Ländern das indirekte Geschäft in den Abschlüssen nicht gesondert ausgewiesen werden muß.

Funktionen der Rückversicherung

Hauptfunktion der Rückversicherung ist der Schutz des Erstversicherers vor starken Schwankungen im kalkulierten Schadenverlauf, die die Rentabilität und im ungünstigsten Falle den Bestand des Unternehmens gefährden könnten. Solche Schwankungen können folgende Ursachen haben:[4]

[3] vgl. Gerathewohl, Klaus u.a., Rückversicherung Band 1, S. 609
[4] Grossmann, Marcel, Rückversicherung S. 26

- Zufallsrisiko
  Das Eintreten von zufallsbedingten Großschäden auf einzelne Risiken, deren Versicherungssumme weit über dem Durchschnitt der insgesamt versicherten Risiken hinausgeht.
- Kumulrisiko
  Das Auftreten von kumulierenden Einzelschäden. Kumule bilden Risiken aus gleichen oder voneinander unabhängigen Versicherungspolicen, die das gleiche Schadenereignis betreffen.
- Änderungsrisiko
  Ansteigen der Schadenhäufigkeit oder des Schadendurchschnitts im allgemeinen. Zum Änderungsrisiko zählen z.B. besonders schlechte Wetterverhältnisse im Winter, die zu einem Ansteigen der Unfallhäufigkeit führen oder aber Änderungen von Risikofaktoren im Zeitverlauf wie z.B. in der Lebensversicherung durch das Auftreten von AIDS.
- Irrtumsrisiko
  Irrtümer bei der Berechnung, fehlende oder überholte Statistiken oder deren falsche Anwendung, die dazu führen, daß sich der Erstversicherer in der Lage sieht das Risiko alleine zu tragen.

Darüber hinaus kann die Rückversicherung den Erstversicherer in die Lage versetzen, seine Zeichnungskapazität im allgemeinen zu erweitern, d.h. mehr Risiken annehmen zu können.

Nicht zu unterschätzen ist auch der Service, den der Rückversicherer dem Erstversicherer bieten kann oder vertraglich geregelt bieten muß.

Eine weitere Funktion ist die Versicherung von besonders großen, mit dem technischen Fortschritt verbundenen Risiken wie z.B. Kernkraftwerken, die ohne Rückversicherung überhaupt nicht versicherbar wären.

Rückversicherungsformen

Unter Berücksichtigung der verschiedenartigen Risikomanifestationen wie Zufall, Kumul-, Änderungs- und Irrtumsrisiko und der verschiedenartigen Branchen wie Sach-, Leben- und Haftpflicht-Unfall-Kraftfahrt-Rückversichung, die ihrerseits weiter untergliedert werden und jeweils versicherungstechnische Besonderheiten aufweisen, haben sich im Laufe der Zeit auch verschiedene Rückversicherungs-Vertragstypen herausgebildet (Abbildung 1-2).

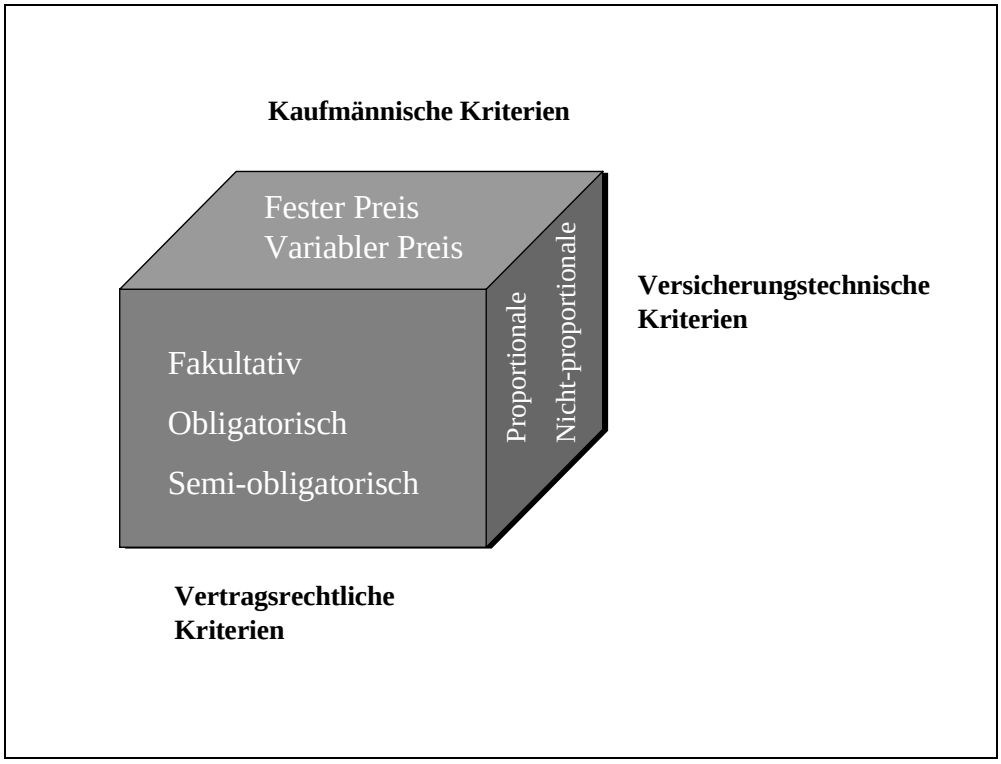


Abb. 1-2: Verschiedene RV-Vertragtypen[5]

Ein Vertrag setzt sich in der Regel aus einer Kombination der drei Kriterien zusammen, die im folgenden näher beschrieben werden, enthält aber von Fall zu Fall unterschiedliche Zusatzklauseln.

Kaufmännische Kriterien

Man unterschiedet zwischen Rückversicherungsverträgen mit variablem oder fixem Preis, je nachdem, ob sich der Preis des Rückversicherungsschutzes nach dem Geschäftsverlauf des Vertrages richtet oder ob der Preis während der gesamten Vertragsdauer fest vereinbart ist.

Vertragsrechtliche Kriterien

- Fakultative-Rückversicherung
  Die fakultative Rückversicherung versichert einzelne Policen. Erstversicherer und

[5] vgl. Grossmann, Marcel, Rückversicherung, S. 76

Rückversicherer sind frei, den angebotenen Rückversicherungsschutz anzunehmen oder abzulehnen. Mit einem speziellen Rahmenvertrag können Bedingungen generell geregelt werden.[6]

- Obligatorische-Rückversicherung
  Bei der obligatorischen Rückversicherung werden alle Risiken eines bestimmten Vertrags oder ein Ausschnitt daraus versichert, z.B. alle in Deutschland direkt gezeichneten Feuerversicherungen oder das französische Transportgeschäft. Der Rückversicherer kann den Rückversicherungsschutz für einzelne Risiken nicht ablehnen und der Erstversicherer nicht einzelne Risiken aus dem Versicherungsschutz herausnehmen. Abweichend davon kann der Erstversicherer im Einzelfall Risikoteile vorweg fakultativ abdecken.

- Semi-Obligatorische-Rückversicherung
  Der Erstversicherer ist frei, dem Rückversicherer bestimmte Risiken anzubieten, während der Rückversicherer zur Annahme verpflichtet ist. In dem Fall spricht man von einem fakultativ-obligatorischen Vertrag oder Open Cover. Der Rückversicherer erhält somit ein breites Sortiment und nicht nur vereinzelt Risiken. Freiheit und Bindung kann auch andersherum verteilt sein. Das kommt in der Praxis aber sehr selten vor.

Versicherungstechnische Kriterien

- Proprotionale-Rückversicherung
  Bei der proportionalen Rückversicherung wird das rückversicherte Risiko zwischen Erstversicherer und Rückversicherer nach einem festen Prozentsatz aufgeteilt, der zugleich den Anteil des Rückversicherers an eingetroffenen Schäden sowie die Prämienverteilung bestimmt[7]. Die drei wichtigsten Vertragsformen werden im folgenden kurz vorgestellt.

  Quoten-Rückversicherung

  Bei der Quoten-Rückversicherung wird der Rückversicherer an dem gesamten Vertragsbestand einer bestimmten Branche oder eines Teils davon, ohne Rücksicht auf die Höhe der Versicherungssummen, prozentual beteiligt.

[6] vgl. Grossmann, Marcel, Rückversicherung, S. 77
[7] vgl. Pfeiffer, Christoph, Einführung in die Rückversicherung, S 45

Summenexzedent-Rückversicherung

Der Rückversicherer wird mit einem festen Prozentsatz (Maximum) an allen Versicherungen beteiligt, die eine bestimmte Versicherungssumme und somit den Selbstbehalt des Erstversicherers überschreiten. In der Regel wird festgelegt, bis zu welchem Betrag der Rückversicherer Schäden übernehmen muß. Der Anteil des Rückversicherers an Schäden und Prämien ergibt sich aus dem Quotienten von Versicherungssumme und Selbstbehalt.

Quotenexzedenten-Rückversicherung

Die Quotenexzedenten-Rückversicherung ist ein Kombination der quoten- und Summenexzedenten-Rückversicherung mit Vorwegquote oder Vorwegexzedent.

- Nicht-proprotionale-Rückversicherung
  Nach Pfeiffer liegt das Wesen der nicht-proportionalen RV gerade darin, daß die Leistung des Rückversicherers ausschließlich durch die Höhe des Schadens bestimmt wird und keine proportionale Aufteilung des einzelnen Risikos und der dafür erhobenen Prämie stattfindet[8]. Die drei wichtigsten der nicht proprotionalen RV werden im folgenden kurz erläutert.

  Schadenexzedenten-Rückversicherung

  Der Schadenexzedent-Rückversicherungsvertrag stellt grundsätzlich auf einzelne Schadenereignisse ab. Es haben sich zwei Formen herausgebildet. Der Einzel-Schadenexzedenten-Rückversicherungsvertrag, auch Working Excess of Loss (WXL) bezeichnet und der Kumul-Schadenexzedenten-Rückversicherungsvertrag auch Catastrophe Excess of Loss (Cat. XL) genannt. Beim WXL entscheidet der Erstversicherer, welchen Betrag er pro Einzelschaden selbst tragen will. Diesen Betrag nennt man Priorität oder erstes Risiko. Alle Beträge, die darüber hinausgehen, betreffen den Rückversicherer bis zu einer Höchstgrenze pro Einzelrisiko. Der Rückversicherer kalkuliert die Prämie, die er zur Deckung der Schäden unter Berücksichtigung eines Kosten- und Gewinnzuschlags benötigt, anhand von Statistiken über den Schadenverlauf und nicht anhand der Originalprämie. Beim Cat. XL verpflichtet sich der Rückversicherer, von der Summe aller Schadenzahlungen des Erstversicherers, die auf dasselbe Schadenereignis zurückzuführen sind, den Teil zu übernehmen, der eine vorher festgelegte Priorität überschreitet. Die Priorität muß in diesem Fall so hoch

[8] vgl. Pfeiffer, Christoph, Einführung in die Rückversicherung, S. 57

bemessen sein, daß im Falle eines Schadens auf ein einzelnes Risiko keine Deckung in Frage kommt.

Jahres-Schadenexzedenten-Rückversicherung

Dieses Vertragsform wird üblicherweise auch Stop Loss oder kurz SL bezeichnet. Beim SL übernimmt der Rückversicherer alle Schäden, die während eines Jahres eine vom Erstversicherer festgelegte Grenze überschreiten, die sich aufgrund des technischen Verlaufs eines bestimmten Versicherungszweiges während eines Jahres ergeben hat. Diese Grenze wird Stop Loss-Punkt bzw. Priorität genannt und meistens in Prozent der Originalprämie ausgedrückt. Auch beim SL wird im allgemeinen die Höchsthaftung begrenzt.

Höchstschaden-Rückversicherung

Dieses Vertragsform wird häufig mit HS-RV abgekürzt. Bei der HS-RV übernimmt der Rückversicherer eine im voraus bestimmte Anzahl der höchsten Schäden, die während eines Geschäftsjahres eingetreten sind, und zwar in der Regel hundert Prozent.

1.5 Das PROfessionelle Rückversicherungs-Informations-System PRORIS

PRORIS ist ein „**PRO**fessionelles **R**ückversicherungs-**I**nformations-**S**ystem“ auf Client/Server Basis. Es deckt sowohl die Anforderungen an die passive Rückversicherung eines Erstversicherers als auch die Anforderungen eines professionellen Rückversicherers an die aktive und passive Rückversicherung ab[PRORIS 95].

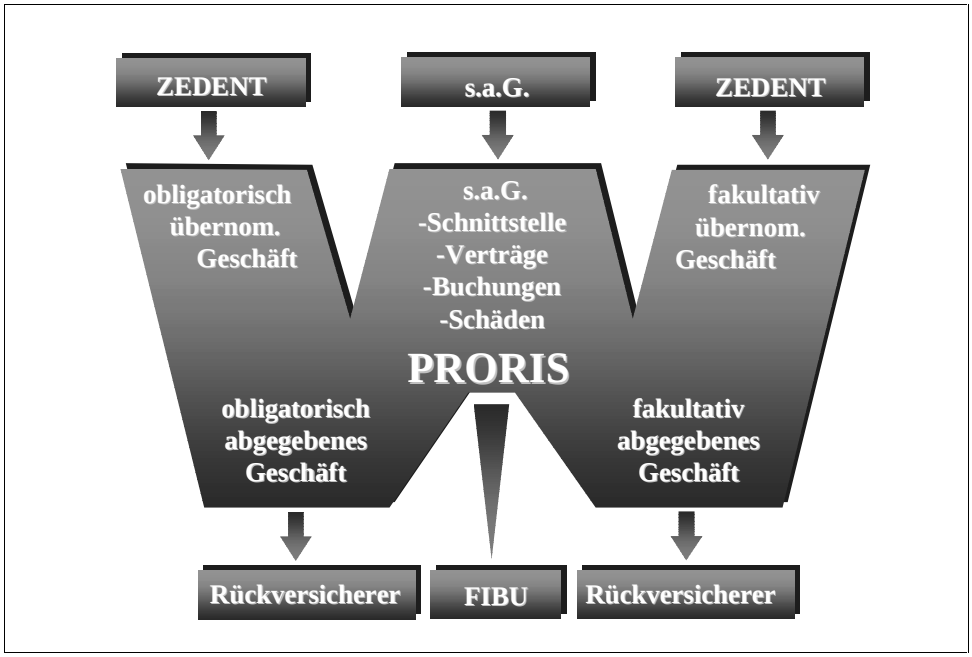


Abb. 1-3: PRORIS-W [PRORIS 95]

Das System ist als zwei Schichten Client/Server-Anwendung mit zentraler Datenhaltung realisiert. Die Anwendungslogik für den PC-Teil ist in den Anwendungsprogrammen von PRORIS implementiert, die im Hauptspeicher der jeweiligen Arbeitsstationen ablaufen. Darüber hinaus gibt es Schnittstellenprogramme, die Daten aus Großrechnersystemen PRORIS-Konform zur Verfügung stellen.

Die Datenhaltung erfolgt zentral in einem relationalen Datenbanksystem. Bei einigen Installationen ist die Datenhaltung verteilt über mehrere Datenbankserver im PC- und Großrechnerumfeld organisiert.

Das System ist im PC-Umfeld in der Programmiersprache C und auf dem Großrechner in COBOL implementiert. Als Abfragesprache für die Datenbank kommt Standard SQL zum

Einsatz. Ablaufumgebung für die Anwendung auf der Arbeitsstation war ursprünglich ausschließlich OS/2 mit dem Presentation Manager von der IBM. Das System wurde 1997 auf Microsoft Windows NT migriert. Randbedingung bei dieser Migration war, daß das System auch weiterhin unter OS/2 lauffähig bleiben mußte. Als relationales Datenbanksystem kommt die DB2-Familie der IBM auf diversen Betriebssystem-Plattformen zum Einsatz. Die Einbindung in das Berechtigungssystem der Netzwerkumgebung ist für IBM LAN Server, Novell Netware und Windows NT Server realisiert.

Das PRORIS-System ist ein sehr umfangreiches und komplexes System. Es besteht aus ca. 15.000 Konfigurationselementen[9] wie z.B. Programmcode, Hilfedateien und SQL-Anweisungen. Die Datenbankstruktur besteht aus ca. 1100 Datenbankelementen wie beispielsweise Tabellen und Sichten.

PRORIS ist aufgrund seiner komplexen Struktur und aufgrund der Tatsache, daß das Produkt einer hohen Änderungshäufigkeit unterliegt, ein besonders gutes Beispiel für die Notwendigkeit eines funktionierenden SKM-Systems.

Das PRORIS-Projektteam setzt sich aus ca. 15 Personen zusammen, die in Rollen wie Projektmanager QM-Mitarbeiter Entwickler tätig sind. Einzelnen Personen fallen durchaus mehrere Rollen zu. Das bedeutet, daß ein Mitarbeiter Entwickler sein kann und gleichzeitig für Entwicklungsergebnisse eines anderen als QM-Mitarbeiter bzw. Tester agiert.

Das PRORIS-System ist bei ca. 15 Kunden im Einsatz und hat ca. 150 Anwender. Hinzu kommen jeweils ca. 2 Mitarbeiter aus den technischen Abteilungen der Kunden, die PRORIS im laufenden Betrieb bzw. bei Releasewechseln betreuen.

[9] zur Begrifflichkeit siehe Abschnitt 2.2.2

2 Software-Konfigurationsmanagement

2.1 Gründe für die Einführung des Software-Konfigurationsmanagements

Softwaresysteme werden heute immer komplexer und bestehen aus einer zunehmend größer werdenden Zahl unterschiedlicher voneinander abhängiger Dateien, Elementen, Komponenten, Teilsystemen etc. Mit dem Aufkommen der Internet-/Intranet-Anwendungen verkürzen sich die Release-Zyklen[10] von bisher Monaten auf Tage oder sogar Stunden. Entsprechend effektiver müssen Kontroll- und Koordinationsmechanismen wirken.

In komplexen Software-Entwicklungsumgebungen müssen Teams oftmals gleichzeitig Teile derselben Software ändern, wodurch wechselseitige Abhängigkeiten auftreten. Diese Tatsache sowie die Komplexität heutiger Client/Server-Architekturen, heterogener Netzwerke und geografisch verteilter Entwicklungsteams verstärken die Notwendigkeit des Einsatzes von SKM-Systemen.

Laut Höft rechtfertigen folgende Probleme den Einsatz eines Software-Konfigurationsmanagements [Höft 85]

- Mengenproblem
  Software-Systeme, beispielsweise in Form umfangreicher betrieblicher Informationssysteme, bestehen aus einer großen Anzahl von Elementen (z.B. Modulen, Datenbankdefinitionen, etc.), zwischen denen es eine große Anzahl von Beziehungen gibt. Diese Elemente gilt es konsistent zu verwalten.
- Änderungsproblem
  Änderungen sind in der Entwicklung und Pflege von Software unvermeidbar. Zusammen mit dem zuvor genannten Mengenproblem sind ungeplante und unkontrollierte Änderungen eine potentielle Quelle von Mängeln und Fehlern, die oft auf nicht aufeinander abgestimmte Software-Elemente zurückzuführen sind.
- Lebensdauerproblem
  Im Lebenszyklus eines Produkts kann es zu Mehrfacheinsätzen oder zu Überlegungen kommen, die Lebensdauer des Produkts zu erhöhen. Dies hat häufig die Aufteilung von Wartungs- und Weiterentwicklungsarbeiten auf unterschiedliche Mitarbeiter oder Organisationseinheiten zur Folge. Die Durchführung der Wartung ist nur durch ausreichende Informationen über die Systemelemente und deren Abhängigkeiten voneinander möglich.

[10] zur Begrifflichkeit siehe Abschnitt 2.2.3

- Konsistenzproblem
  Systeme, die aus einer Vielzahl von Elementen bestehen, lassen sich nur schwer konsistent halten. Ein Element liegt in verschiedenen Zuständen vor. Beispielsweise existiert von einem Baustein zunächst eine Modulspezifikation, dann wird ein Entwurf für diesen Baustein angefertigt und schließlich der Programmcode erzeugt. Dazu kommen Testfälle, Testdaten und eine Teststrategie, die zur Validierung dieses Bausteins dienen. Das Konsistenzproblem wird zusätzlich dadurch verschärft, daß aufgrund von Anforderungen das System weiterentwickelt wird bzw. unterschiedliche Ausprägungen einzelner Elemente parallel verwaltet werden müssen, da z.B. unterschiedliche Relaesestände[11] der Software im produktiven Einsatz sind.

Einige typische Gründe bzw. Fragestellungen für die Einführung eines Software-Konfigurationsmanagement sind folgende:

- Bereits korrigierte Fehler treten wieder auf.
- Es ist unklar, welche Korrekturen warum durchgeführt wurden.
- Es ist unklar, ob ein Fehler bereits behoben wurde.
- Wie stellt man eine Konfiguration her, die alle Fehlermeldungen bis zu einem bestimmten Termin umfaßt.
- Die letzten Verbesserungen waren fehlerhaft und müßten entfernt werden.
- Wie unterscheidet sich eine neue Bezugskonfiguration[12] von der alten ?
- Welchen Stand hat die Software und woraus besteht dieser Stand ?

Die geschilderten Probleme zeigen, daß adäquate Hilfsmittel zu ihrer Bewältigung bereitgestellt werden müssen. Erfolgt dies nicht, kommt es zwangsläufig zu erheblichen Qualitätsminderungen bei der Entwicklung, Pflege und Einsatz von Software.

2.2 Begriffe des Software-Konfigurationsmanagements

2.2.1 Konfiguration, Software-Konfiguration, Software-Konfigurationsmanagement

Das Konfigurationsmanagement von Software ist die Gesamtheit der Verfahren zur eindeutigen Kennzeichnung der Konfiguration eines Software-Systems zu bestimmten

[11] zur Begrifflichkeit siehe Abschnitt 2.2.3
[12] zur Begrifflichkeit siehe Abschnitt 2.2.5

Zeitpunkten des Software-Lebenszyklus mit dem Zweck, alle Änderungen dieser Konfiguration systematisch zu überwachen, die Konsistenz des Software-Systems sicherzustellen und jederzeit die Rückverfolgung anzubieten. Zuerst ist zu klären, was unter Konfiguration zu verstehen ist.

In der EN ISO 10007 wird der Begriff Konfiguration wie folgt definiert [ISO 10007]:

> „Funktionelle und physische Merkmale eines Produkts, wie sie in seinen technischen Dokumenten beschrieben und im Produkt verwirklicht sind. Insbesondere kann sich der Begriff Konfiguration sowohl auf Hardware- als auch auf Software-Systeme beziehen."

Die Firma Siemens definiert den Begriff folgendermaßen:

> „Eine Konfiguration ist eine benannte und formal freigegebene Menge von Entwicklungsergebnissen, die in ihrer Wirkungsweise und ihren Schnittstellen aufeinander abgestimmt sind und gemeinsam eine vorgegebene Aufgabe erfüllen sollen."

Zusammenfassend läßt sich unter Software-Konfiguration die Gesamtheit der Software-Elemente bezeichnen, die zu einem bestimmten Zeitpunkt im Lebenszyklus des Software-Produkts in ihrer Wirkungsweise und ihren Schnittstellen aufeinander abgestimmt sind. Ein Software-Element ist das „Atom" der Konfigurationsverwaltung, also der kleinste für diesen Zweck als unteilbar behandelte Bestandteil des Software-Produkts, z.B. ein Dokument oder ein Modul. Als unteilbar behandelt bedeutet, daß das Element als Ganzes eindeutig gekennzeichnet und dem Änderungsdienst[13] unterworfen ist.

Konfigurationen dienen folgenden Zwecken [Wallmüller 90]:

- Sie regeln die Zugehörigkeit von Entwicklungsergebnissen zu Teilsystemen oder Systemen.
- Sie bringen Ordnung in die Zustandsvielfalt von Entwicklungsergebnissen.
- Sie bilden Bezugspunkte für definierte Entwicklungsschritte, da sie die Ergebnisse vorangegangener Entwicklungsschritte festhalten und somit eine wohldefinierte Basis darstellen.

[13] siehe Abschnitt 2.3.2

- Sie halten Zwischenergebnisse eines Produkts über seine gesamte Lebensdauer fest und gewährleisten dadurch die Wiederverwendbarkeit und Wartbarkeit des Produkts.
- Sie bilden eine Informationsquelle zur Analyse des Entwicklungsprozesses und des Produkts.

Unter Softwarekonfigurationsmanagement versteht man abschließend die Gesamtheit von Methoden, Werkzeugen und Hilfsmitteln, die die Entwicklung und Pflege eines Software-Produkts als eine Folge von kontrollierten Änderungen (Revisionen[14]) und Ergänzungen (Varianten[15]) an gesicherten Prozeßergebnissen unterstützen.

2.2.2 Konfigurationselemente

Unter einem Konfigurationselement (KE) versteht man eine elementare Einheit einer Konfiguration mit einer eindeutigen Identifikation. Es enthält somit keine Untereinheiten die unabhängig voneinander variieren können. Ein KE kann gleichzeitig in mehreren Revisionen und Varianten existieren.

Unter KE versteht man nicht nur die Programmcode, sondern auch alle anderen Projektdokumente sowie umgebungsbeeinflußende Faktoren wie z.B. Compiler, Linker, Bibliotheken etc.

Hier einige Beispiele für KE:

- Pflichtenheft
- Produktmodell
- Entwurfsdokumentation
- Programmcode
- Compiler, Linker, Hilfegenerator
- Einstellungen für Werkzeuge
- Testplan
- Testdaten
- Benutzerhandbuch

[14] zur Begrifflichkeit siehe Abschnitt 2.2.4
[15] zur Begrifflichkeit siehe Abschnitt 2.2.4

- Projektpläne

KE lassen sich nach der Entstehungsart klassifizieren. Es gibt Quellelemente und abgeleitete Elemente, die sich nach eindeutigen reproduzierbaren Regeln aus Quellelementen, teilweise über mehrere Stufen, ableiten lassen. Abgeleitete Elemente können somit ihrerseits wiederum Quellelemente für andere Elemente sein.

2.2.3 Release, Version, Patch

Die Begriffe Release, Version und Patch beziehen sich in der Regel auf auslieferbare Teile des Softwareprodukts, die nach festgelegten Numerierungsregeln bestimmt werden. Ein einfaches Beispielschema für eine Numerierungsregel ist ein Identifikator, der sich wir folgt zusammensetzt:

Identifikator = Release.Version.Patch

Diesem Beispiel zur Folge würde der Identifikator 1.2.1 das Release 1 in der Version 2 mit dem Patch 1 bezeichen. Die Releasenummer steht in diesem Beispiel für wesentliche Änderungen oder Neuentwicklungen des betrachteten Softwareprodukts. Die Versionsnummer bildet in diesem Zusammenhang laufende Änderungen und Verbesserungen des Systems ab, die aber keine wesentlichen Erweiterungen darstellen. Die Patchnummer trägt der Tatsache Rechnung, daß in dem Softwareprodukt Fehler auftreten, die nicht erst mit der nächsten regulären Auslieferung des Produkts behoben werden dürfen. Ein Patch ist somit eine Fehlerkorrektur oder Änderung die eine außerordentliche Auslieferung verursacht. Varianten sowie die einzelnen Änderungen an KE sind dabei noch nicht berücksichtigt[16].

Bei Windows NT von Microsoft Corporation z.B., ist für den Außenstehenden ein stringentes Numerierungsschema nicht erkennbar. Das erste marktrelevante Release war Windows NT 3.51. Diese Releasenummer orientierte sich an der damaligen Windowsversion für den Heimanwender Windows 3.11. Der Wechsel auf das Release Windows NT 4.0 enthielt wesentliche Änderungen und Funktionserweiterungen des Systems und rechtfertigt somit, nach oben genannten Schema, ein Wechsel in der

[16] siehe auch Abschnitt 2.2.4 Revisionen und Varianten

Releasenummer. Alle weiteren Änderungen an Windows NT 4.0 wurden in Form von Service Packs ausgegeben. Diese Service Packs haben sowohl Fehlerkorrekturen beinhaltet als auch wesentliche Erweiterungen des Systems enthalten. Darüber hinaus gibt es auf bestimmten Service Packs aufsetzend Fixpacks für spezifische Probleme wie z.B. das „Jahr 2000 Problem“. Dieses Beispiel zeigt, daß die Bezeichnungen der Releasestände von Softwareprodukten häufig aus Marketinggesichtspunkten vergeben werden und nicht die interne Struktur der tatsächlichen Änderungshistorie darstellen.

2.2.4 Revisionen und Varianten

Revisionen stellen die Veränderungen an Konfigurationseinheiten einer Entwicklungslinie dar. Varianten bezeichnen dagegen die unterschiedlichen Entwicklungslinien der Konfigurationseinheiten. Bestimmte Entwicklungslinien können lediglich temporären Charakter haben und zu einem späteren Zeitpunkt mit der Hauptentwicklungslinie wieder zusammengeführt werden.

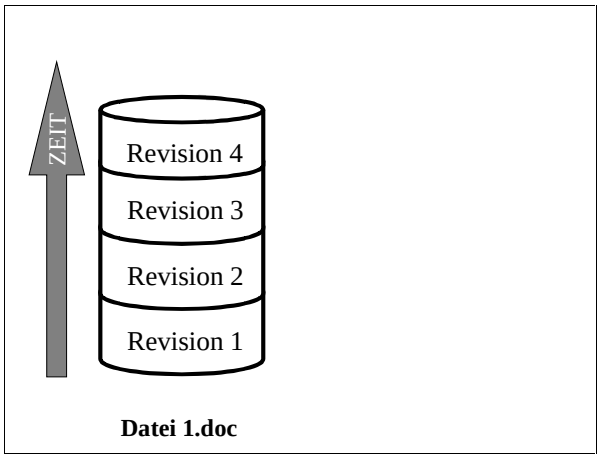


Abb. 2-1: Revisionen

Ein nützliches Konzept um die Begriffe Variante und Revision einzuführen, sind zusammenhängende Versionsgruppen von KE. Eine Versionsgruppe ist eine Menge von KE, die durch die Relation „ist Variante von“ und „ist Revision von“ verbunden sind. Die Relation „y ist Variante von x“ kann folgendermaßen definiert werden[Tichy 88]:

> „x und y sind Konfigurationseinheiten, die existieren und eine wesentliche Eigenschaft gemeinsam besitzen. Eine wesentliche Eigenschaft kann z.B. dieselbe

funktionale Spezifikation eines Moduls bedeuten. Die Implementierungen dieses Moduls sind hingegen verschieden.“

Ein Beispiel für den Begriff „Variante“ ist ein Software-Werkzeug, das für verschiedene Betriebssysteme entwickelt werden soll. Die funktionalen Anforderungen an das Werkzeug sind für die unterschiedlichen Betriebssysteme die gleichen. Die Implementierung unterscheidet sich jedoch aufgrund unterschiedlicher Betriebssystemschnittstellen.

Die Relation „y ist Revision von x“ kann folgendermaßen festgelegt werden:

„x und y sind KE und y wurde durch Änderung einer Kopie von x erzeugt.“

Dahinter steht die Absicht, daß y eine Verbesserung von x ist und x ersetzt.

2.2.5 Bezugskonfiguration

Das wesentliche Ziel des Software-Konfigurationsmanagements ist die effiziente Verwaltung der Software-Konfigurationen im Lebenszyklus des Produkts. Ein fundamentales Konzept dazu ist jenes der Bezugskonfiguration auch als Baseline oder Freigabe bezeichnet.

Diese definieren zum einen die einzelnen Bestandteile (KE), zum anderen die Informationen dazu, wie und in welcher Umgebung aus diesen Einzelteilen das Endprodukt wieder hergestellt werden kann. Im einzelnen sind das folgende Informationen

- KE in bestimmter Revision und ggf. auch Variante
- Werkzeuge mit Versionsstand wie z.B. Make, Compiler, Pre-Compiler, Librarys etc.
- Parametrisierung der Werkzeuge
- Plattformkonfiguration mit Versionsständen und Parametrisierung
    - z.B. Entwicklungsplattform, Zeilplattform, Testplattform

Jede Änderung an einem der Elemente führt zu einer neuen Konfiguration. Dies zeigt, daß auch die Umgebung des Softwareentwicklungsprozesses im Konfigurationsmanagement Berücksichtigung finden muß.

In Abbildung 2-2 sieht man, wie Bezugskonfigurationen z.B. mit Hilfe von Revisionen festgehalten werden, die per Versionskennzeichen (Baseline 1 und Baseline 2) miteinander verbunden werden.

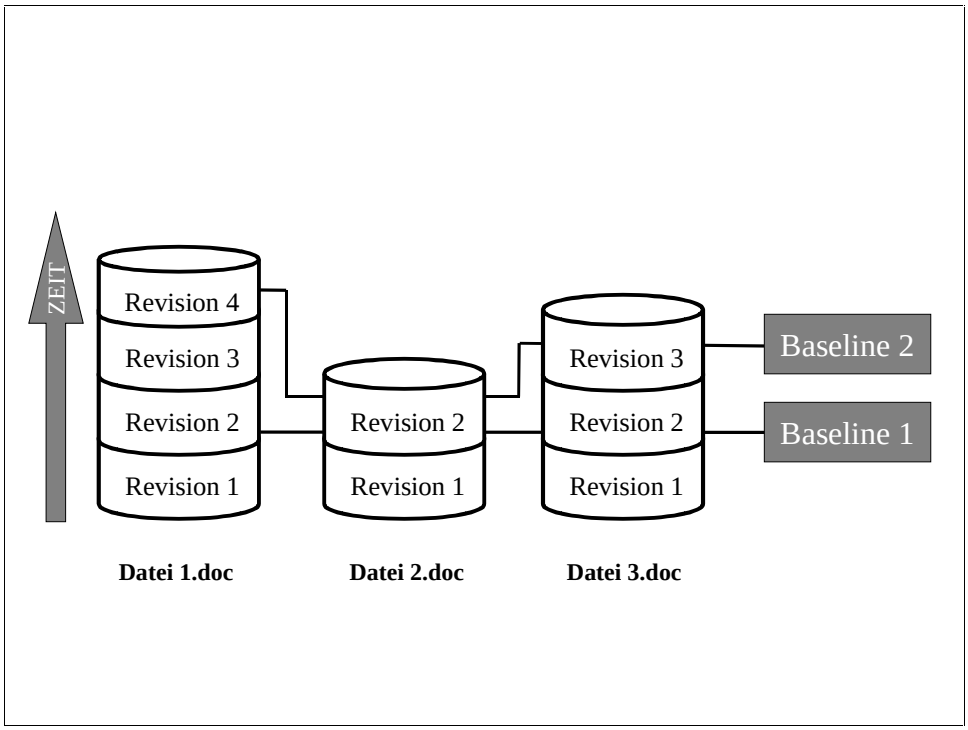


Abb. 2-2: Bezugskonfigurationen

Die EN ISO 10007 definiert Bezugskonfiguration wie folgt:

> "Die formell zu einem bestimmten Zeitpunkt festgelegte Konfiguration eines Produkts, die als Grundlage für weitere Tätigkeiten dient."

Bezugskonfigurationen sollten immer dann festgelegt werden, wenn es während des Lebenslaufs des Produkts nötig ist, seine Konfiguration so festzulegen, daß sie als Ausgangspunkt für weitere Aktivitäten dienen kann. Die Notwendigkeit eine Bezugskonfiguration festzulegen hängt insofern von der Projektorganisation bzw. vom Vorgehensmodell ab.

Bersoff, Henderson und Siegel gehen von einem phasenorientierten Modell aus, bei dem am Ende einer jeden Entwicklungsphase eine Bezugskonfiguration freigegeben wird. Sie unterscheiden dabei Anforderungs-, Konzept-, Entwurfs-, Integrations- und Produktkonfiguration [Bers 80].

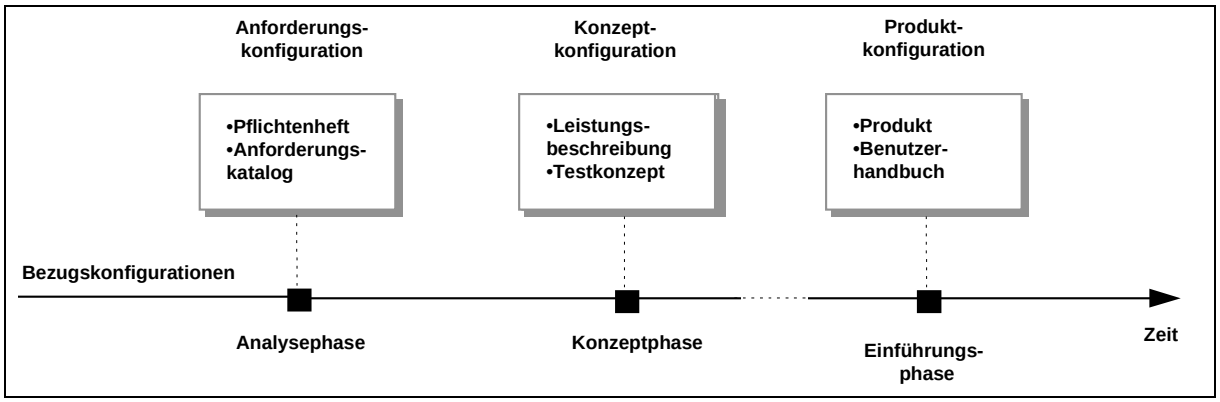


Abb. 2-3: Bezugskonfiguration nach Bersoff, Henderson und Siegel

Bei einer versionsorientierten revolvierenden Projektorganisation und -planung stellen die einzelnen Systemversionen die Bezugskonfigurationen dar, von denen ausgehend die Ziele für die jeweils nächste Systemversion geplant und festgelegt werden.

Wesentliche Aktivitäten bei der Erstellung und Pflege von Bezugskonfigurationen sind die Planung, Durchführung und Aufzeichnung aller Änderungen mit dem Ziel, daß die aktuelle Konfiguration mit ihrem Änderungsstatus zu jedem Zeitpunkt bekannt ist. Dies setzt voraus, daß alle Bezugskonfigurationen definiert werden und die Elemente jeder Bezugskonfiguration identifiziert und benannt sind.

Der Änderungsprozeß umfaßt die Erfassung und Berichterstattung über Änderungen sowie die Verifikation einer oder mehrerer Bezugskonfigurationen (Abbildung. 2-4). Die Verifikation erfolgt durch technisch orientierte Reviews oder Audits.

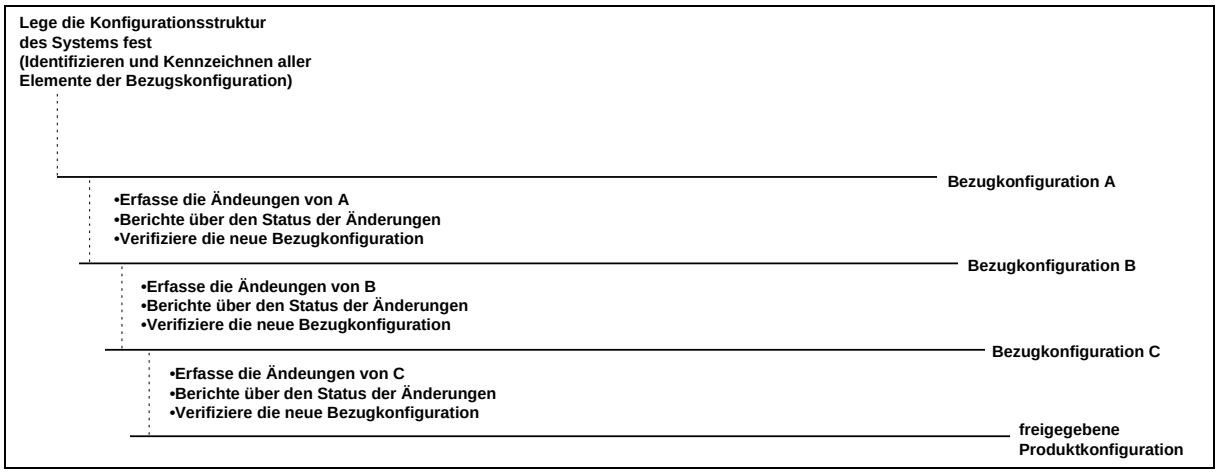


Abb. 2-4: Änderungswesen mit Bezugskonfigurationen [IEEE 88]

Die Entwicklung und Pflege eines Produkts erfolgt also durch eine Reihe geplanter und freigegebener Bezugskonfigurationen. Sie werden häufig von der Projektleitung und dem Wartungsverantwortlichen als Meilensteine behandelt. Das Produkt selbst besteht aus einer großen Anzahl von Software-Elementen, die die Konfigurationselemente der einzelnen Bezugskonfigurationen bilden.

2.3 Aufgaben des Software-Konfigurationsmanagements

Software ist relativ einfach modifizierbar. Das Löschen oder Modifizieren von Programmteilen kann katastrophale Folgen nach sich ziehen, wenn nicht systematisch geplant und geprüft wird [Scheller 96].

Die Problematik des Änderns ist in der Wartung besonders groß. Vielfach werden Änderungen nur unvollständig realisiert und dokumentiert. Das Problem wird noch dadurch verschärft, daß die Dokumentation teilweise oder ganz fehlt und nicht aktuell ist. Der Stand der Software-Elemente zum Zeitpunkt der Freigabe ist nicht immer eindeutig reproduzierbar. Die Auswirkungen dieses Zustands sind nicht mehr kontrollierbar und führen häufig früher als nötig zur Notwendigkeit einer Neuentwicklung der Systeme

Die Grundidee des Software-Konfigurationsmanagements besteht darin, mit Disziplin und Systematik Ordnung in den KE zu halten. Dabei sind folgende Aufgaben zu erfüllen:

- Die erste Aufgabe des SKM ist die Konfigurationsidentifizierung. Dabei geht es um die Bestimmung der für eine Konfiguration relevanten Elemente wie z.B. Module, Hilfetexte, Datenbankdefinitionen. Damit diese Elemente eindeutig identifiziert werden können, ist es notwendig Numerierungsregeln aufzustellen.
- Bei der zweiten Aufgabe geht es darum, alle Änderungen an einer Konfiguration zu überwachen. Dazu werden alle Änderungsanstöße in Form von Meldungen erfaßt und von einem einzurichtenden Änderungsausschuß einer Prüfung unterzogen, der die Notwendigkeit der Änderung untersucht.
- Die dritte Aufgabe besteht darin, alle Änderungen zu erfassen und aufzuzeichnen. Diese Konfigurationsbuchführung ermöglicht es, jederzeit Auskunft darüber geben zu können, welche Änderungen bisher am System durchgeführt wurden.
- Die vierte Aufgabe ist das Konfigurationsaudit. Dabei geht es darum, daß die am Beginn der Systementwicklung aufgestellten Anforderungen der Benutzer und die

Anforderungen, die während der Systementwicklung an einzelne Systembausteine gestellt worden sind, auch nach den Änderungen erfüllt bleiben.

2.3.1 Konfigurationsidentifizierung

Bei der Konfigurationsidentifizierung geht es zuerst um die Auswahl und Strukturierung der Konfigurationseinheiten. Dabei wird das gesamte Produkt in logisch zusammenhängende und untergeordnete Einheiten eingeteilt. Bei der Auswahl der Konfigurationseinheiten werden alle relevanten Dokumente - die zur Weiterverarbeitung möglichst in maschienenlesbarer Form vorliegen sollten - und deren Beziehungen zueinander berücksichtigt. Dies können sowohl wenig formalisierte Dokumente wie z.B. Verträge, Änderungswünsche, Protokolle und Konzepte als auch für stark formalisierte Dokumente wie z.B. Datenbankdefinitionen oder Progammcode sein. Darüber hinaus können auch Richtlinien für die Kennzeichnung maschienenlesbarer Elemente (Programme und Bibliotheken) festgelegt werden, die die Beziehungen zu den Quellenelementen herstellen und damit eine Rückverfolgung ermöglichen. Der Detaillierungsgrad, mit dem ein Produkt dem Konfigurationsmanagement unterzogen wird, hängt vom Grad der gewünschten Überwachung und den gewünschten Einflußmöglichkeiten im Entwicklungsprozeß ab.

Für die Identifizierung der Konfigurationseinheiten werden Namenskonventionen und Numerierungsregeln aufgestellt. In den Namenskonventionen kann z.B. die Produktstruktur Berücksichtigung finden. Die Beziehungen der Konfigurationseinheiten können somit oft allein anhand des Namens nachvollzogen werden. Dadurch wird eine einfachere weniger fehleranfällige Verwaltung der Konfigurationseinheiten sichergestellt. Die Numerierungsregeln dienen der Dokumentation von Änderungen an Konfigurationseinheiten. Ihre Struktur und deren Komplexität hängt von der Anforderungen und der Organisation des Produkts bzw. Projekts ab. Bei umfangreichen Produkten mit vielen Kunden besteht die Notwendigkeit mehrere voneinander temporär unabhängige Entwicklungslinien zu pflegen. Diese Entwicklungslinien werden meist von verschiedenen Arbeitsgruppen, die teilweise an verschiedenen Orten arbeiten, gepflegt. Hinzu kommen unterschiedliche Releasestände, die bei Kunden im Einsatz sind und gepflegt werden müssen. Dieser Umstand bedingt häufig die Notwendigkeit vorläufiger Änderungen (Patches), die schwerwiegende Fehler vor dem nächsten Release beheben. Diese Patches müssen kontrolliert eingebracht werden und müssen somit bei den Numerierungsregeln berücksichtigt werden. Damit diese komplexen Produktstrukturen verwaltet werden können, müssen die Numerierungsregeln ebenfalls Revisionen und Varianten unterscheiden können.

Eine weitere Aufgabe der Konfigurationsidentifizierung ist das Festlegen von Verfahren und Richtlinien zur Bestimmung von Bezugskonfigurationen sowie die Regelung des Freigabeverfahrens. Zu bestimmten Zeitpunkten werden durch förmliche Vereinbarungen Bezugskonfigurationen eingerichtet und als Beginn der formalen Fortschreibung der Konfiguration verwendet. Die Bezugskonfiguration zusammen mit ihren freigegebenen Änderungen bilden dann die aktuelle vereinbarte und somit gültige Konfiguration. Das Freigabeverfahren legt fest, durch welche Organisationseinheiten eine Änderung geprüft werden muß und welche Testverfahren angewendet werden müssen, damit für die Änderung eine Freigabe erfolgt.

2.3.2 Konfigurationsüberwachung/-steuerung

Nach der erstmaligen Freigabe von Konfigurationsdokumenten werden alle Änderungen überwacht und verfolgt. Voraussetzung dafür sind formalisierte Änderungsanträge, die entweder vom Auftraggeber oder aus dem Projektteam initiiert werden. Die Änderungsanträge sollten mindestens die folgenden Informationen enthalten:

- Name des Antragstellers und Erstellungsdatum
- Änderungsgrund (Fehlerbeschreibung, Änderungswunsch)
- Dringlichkeit

Zusätzlich sollten die Änderungsanträge eindeutig identifiziert werden, um eine Rückverfolgbarkeit und das Wiederauffinden zu erleichtern. Es bietet sich an, die einzelnen Schritte im Änderungsprozeß ebenfalls auf dem Änderungsantrag zu dokumentieren, damit der Stand der Änderungsbearbeitung sowie Entscheidungen und Verfügungen, die zu dem Änderungsstand geführt haben, einfach nachvollzogen werden können.

Über die genannten Informationen hinaus können im Rahmen der Konfigurationsbuchführung weitere für das Management und die Entwicklungsprozeßsteuerung wichtige Informationen aufgezeichnet werden, die in regelmäßigen Abständen ausgewertet werden und der Prozeßoptimierung dienen können. Mit Hilfe solcher Informationen ist es z.B. möglich, Aussagen über die Fehleranfälligkeit bestimmter Programmkomponenten oder die Effektivität von Freigabeverfahren zu treffen. In welchem Umfang und Detaillierungsgrad solche Informationen aufgezeichnet werden, hängt von den gewünschten Auswertungs- und Steuerungsmöglichkeiten ab. Der recht aufwendige Prozeß der Datensammlung und Auswertung wird meist nur in größeren Softwareentwicklungsprojekten Anwendung finden [Whitgift 91].

Eine weitere Aufgabe der Konfigurationsüberwachung ist das Festlegen der Organisation und Verfahren zur Bewertung und Genehmigung von Änderungen. Eine in der Praxis häufig vorzufindende Organisationsform ist das Change-Control-Board (CCB). Dies ist ein zentrales Entscheidungsgremium für die Genehmigung, Zurückstellung oder Ablehnung von Anforderungen bei Neuentwicklungen und Änderungsanforderungen während des Projektverlaufs. Die Mitglieder dieses Gremiums werden normalerweise vom Projekteiter bestimmt. Dort wo vertragliche Vereinbarungen die Einbindung des Kunden in den Entscheidungsprozeß verlangen, kann dieser ebenfalls Mitglieder benennen. Auf Basis entscheidungsreifer Vorschläge entscheidet das CCB:

- welche Anforderungen und Änderungswünsche in den aktuellen oder in den zukünftigen Entwicklungsprozeß eingebracht werden,
- welche Fehlermeldungen nicht mehr im aktuellen Entwicklungsprozeß berücksichtigt werden sollen bzw.
- welche Fehlermeldungen unbedingt vor dem nächsten Release als vorläufige Korrektur (Patch) ausgeliefert werden sollen,
- welche Konfigurationen zur Integration oder nach erfolgreichem Systemtest zur Auslieferung an den Anwender freigegeben werden.

Eine Voraussetzung für die Bewertung von Änderungswünschen ist ein Verfahren zur Einstufung der Dringlichkeit. Grundlage für ein solches Klassifizierungssystem können Änderungsauswirkungen, Kundenforderungen, sowie die betroffene Bezugskonfiguration sein.

Besondere Aufmerksamkeit muß den Sonderfreigaben geschenkt werden, die z.B. durch schwerwiegende Fehler in älteren Versionen notwendig werden können. Da Sonderfreigaben nicht wie andere Versionen geplant werden können, sondern einen akuten Problemfall außerhalb der üblichen Releaseplanung darstellen, müssen für sie besondere Verfahren eingerichtet werden. Sie führen z.B. zu Varianten die gegebenenfalls pro Kunde unterschiedlich ausfallen können.

Eine weitere Aufgabe der Konfigurationssteuerung ist die Festlegung der Methoden zur Implementierung von genehmigten Änderungen in den betroffenen Konfigurationseinheiten (Dokumentation, Programmcode, Objektcode), wobei der Änderungsablauf abhängig von der Projektphase unterschiedlich sein kann. Dazu ist es notwendig, Software-Konfigurationseinheiten in verschiedenen Umgebungen zu verwalten, die sich unter Kontrolle unterschiedlicher Personen befinden. Diese Aufgabe wird

Umgebungs-Management genannt. Umgebungsmanagment sollte den Entwicklern eine konsistente, flexible und reproduzierbare Entwickungsumgebung zur Verfügung stellen, die für die unterschiedlichen Aufgaben wie z.B. Fehlerkorrekturen oder Neuentwickungen die jeweils passenden Revisionen und Varianten der Konfigurationseinheiten bereitstellen. In großen Projekten müssen auf der einen Seite Entwickler von den noch unfertigen Ergebnissen anderer Entwickler unbeeinflußt bleiben, auf der anderen Seite müssen sie dazu in der Lage sein, z.B. im Rahmen des Modultests bestimmte Änderungen anderer Entwickler mit einzubeziehen. Um dieses erreichen zu können, muß das Umgebungsmanagement eine detaillierte Kontrolle über gemeinsame und voneinander getrennte Sichten auf die Konfigurationseinheiten ermöglichen.

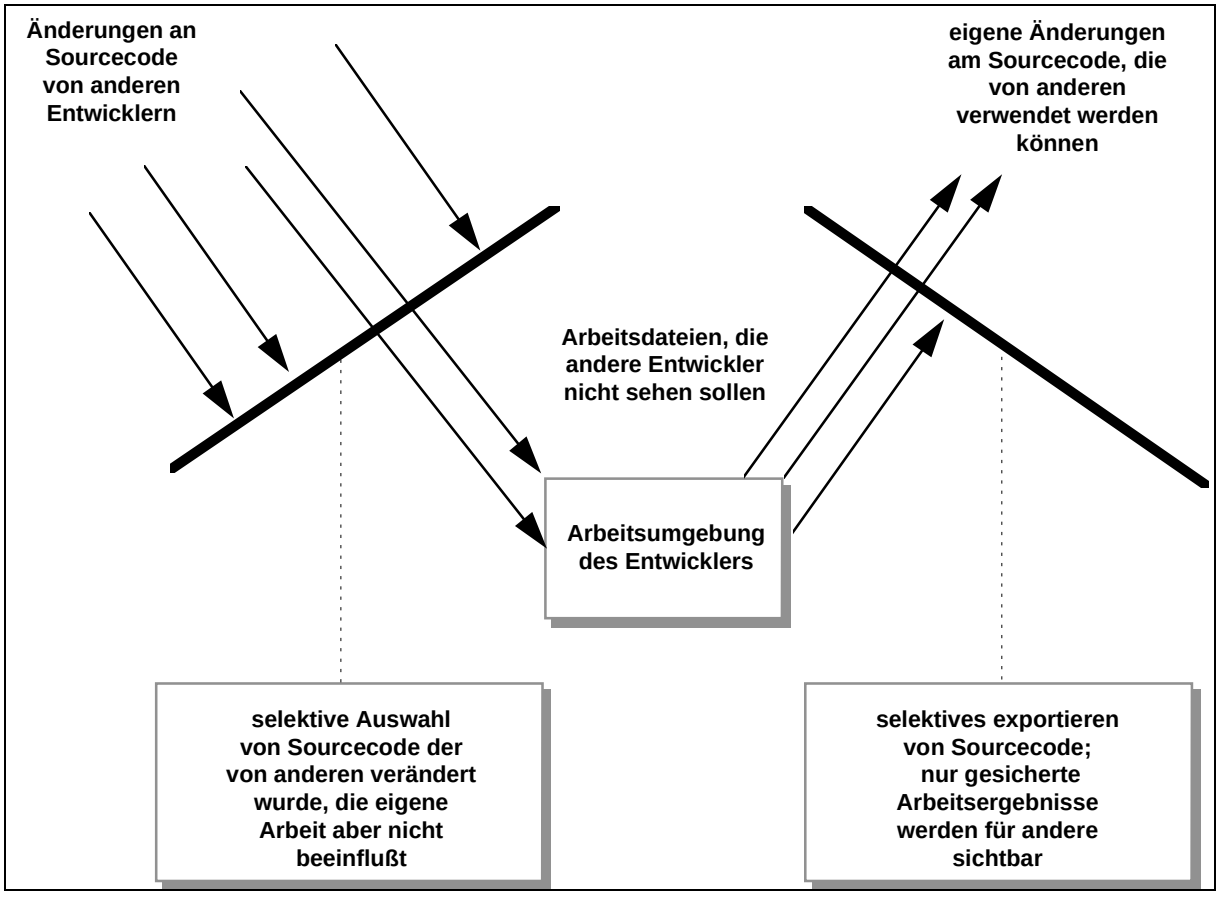


Abb. 2-5: Anforderungen an eine Entwicklungsumgebung

Wenn ein Entwickler z.B. eine ältere Version warten soll, muß er in der Lage sein, auf die alten Programmdateien, Binärdateien, Dokumentationen, Tools und Testverfahren zurückzugreifen. Bei datenbankgestützten Anwendungssystemen kommt die Schwierigkeit hinzu, auch die entsprechende Datenumgebung mit der alten Datenbankstruktur und einem

passenden Testdatenbestand zur Verfügung zu stellen. Wenn die für die Aufgabe passende Umgebung zur Verfügung steht, können die Änderungen auf einer nachvollziehbaren Basis durchgeführt werden. Schließlich werden aus den geänderten Konfigurationseinheiten (z.B. Programmcode) andere Konfigurationseinheiten wie z.B. Objektdateien oder Binärdateien in einem oft mehrstufigen Übersetzungsprozeß erstellt.

Das Build-Management (Systemgenerierung) beschäftigt sich mit der Definition der Regeln und Abhängigkeiten der einzelnen Konfigurationseinheiten, die den Übersetzungsprozeß verläßlich und reproduzierbar gestalten. Dabei müssen einige wesentliche Punkte beachtet werden [Compton 94]:

- abgeleitete Konfigurationseinheiten wie z.B. Binärdateien müssen aus den korrekten Revisionen der Programmcodedateien mit den korrekten Werkzeugen und den korrekten Übersetzungsparametern erstellt werden. Für die korrekte Revision des Programmcodes ist das Umgebungsmanagement zuständig. Aber auch die Werkzeuge und ihre Einstellungen zur Erstellung der Programmkomponenten, Hilfesysteme oder Datenbankdefinitionen, sollten der Konfigurationskontrolle unterliegen und somit eine verläßliche Basis zur Erstellung von abgeleiteten Konfigurationselementen bilden. Das Build-Management wiederum muß in der Lage sein, Änderungen an der Umgebung, unabhängig davon welche Umgebungskomponenten betroffen sind, zu erkennen. Konkret bedeutet das, daß z.B. Änderungen an den Kompilereinstellungen genauso zu einem erneuten Übersetzen führen müssen, wie die Änderung am Programmcode selber.
- abgeleitete Konfigurationseinheiten, die von einem Entwickler erstellt wurden müssen nicht unbedingt mit abgeleiteten Konfigurationseinheiten anderer Entwickler in Konflikt stehen. Insofern sollten die Entwickler in der Lage sein, die abgeleiteten Konfigurationseinheiten anderer nutzen zu können. Das kann sowohl Plattenplatz als auch Zeit sparen.
- das System sollte in der Lage sein zu erkennen, welche Konfigurationseinheiten erneut übersetzt werden müssen und welche nicht, so daß möglichst nur die minimal notwendige Anzahl von Konfigurationseinheiten übersetzt werden muß, um ein konsistentes System wiederherzustellen.
- nachdem die abgeleiteten Konfigurationseinheiten hergestellt wurden, muß es möglich sein, diesen Prozeß anhand einer verläßlichen Liste von Konfigurationseinheiten, die in den Übersetzungsprozeß eingeflossen sind, nachzuvollziehen. Die Liste sollte so genau sein (KE-Name, Revision, etc.), daß die Umgebung zur Übersetzungszeit einfach reproduziert werden kann.

Um die Integrität der Konfiguration zu gewährleisten und die Grundlage für die Änderungsüberwachung zu liefern, ist es wichtig, Konfigurationseinheiten in einer Umgebung zu halten die [DGQ 12-51]:

- sie vor unberechtigter Änderung oder Beschädigung bewahrt,
- die Wiederherstellung im Katastrophenfall gewährleistet,
- die gezielte Wiedergewinnung einer Kopie des überwachten Originals einer bestimmten Revision und Variante erlaubt,
- das Erreichen der Übereinstimmung zwischen dem tatsächlichen Zustand der Konfiguration und dem Design unterstützt.

Die Daten, die im Rahmen des Konfigurationsmanagement gesammelt werden, sind für das Software-Projekt lebenswichtig. Daher müssen an die Datensicherheit und die Datensicherungsverfahren die gleichen hohen Maßstäbe wie an andere kritische Unternehmensdatenbestände angelegt werden.

2.3.3 Konfigurationsbuchführung

Voraussetzung für eine richtige Konfigurationsbuchführung ist eine einwandfreie Identifizierung und Änderungsüberwachung. Die Konfigurationsbuchführung zeichnet alle Informationen für das Management und für die Verwaltung des Konfigurationsmanagementprozesses und der damit zusammenhängenden Tätigkeiten auf und berichtet diese. Sie beginnt mit dem Vorhandensein der ersten Konfigurationseinheiten und dauert über die gesamte Lebensdauer des Produkts fort. Ein wichtiger Faktor für die Qualität der aufgezeichneten Informationen ist ihre zeitnahe Erfassung. Wenn z.B. von einem Entwickler für einen Fehler eine Umgehungsmöglichkeit gefunden wurde, ist es wichtig diese Information auch den anderen Projektmitgliedern - besonders dem Kundensupport - zur Verfügung zu stellen und dies zu tun ohne große Hürden überwinden zu müssen, die dazu führen würden, daß die Aufgabe auf einen späteren Zeitpunkt verschoben und damit eventuell vergessen würde. Dies muß durch eine angemessene Infrastruktur ermöglicht werden, die die Projektmitarbeiter dazu motiviert, derartige Informationen zu hinterlegen. Die Qualität der Konfigurationsbuchführung hängt folglich wesentlich davon ab, daß die Infrastruktur in der Entwicklungsumgebung die Ablauforganisation einfach und effizient unterstützt.

Bei den aufgezeichneten Informationen handelt es sich sowohl um die Konfigurationseinheiten selber, als auch um Informationen über die Konfigurationseinheiten, deren Umgebung, den Beziehungen zwischen den Konfigurationseinheiten, den Beziehungen zu Bezugskonfigurationen und den

Beziehungen zu Änderungsanträgen. Um die benötigten Berichte bereitzustellen, sollten die Daten in einer Weise aufgezeichnet werden, die möglichst vielseitige Auswertungsmöglichkeiten zur Verfügung stellt. Typische Berichte sind z.B.:

- Aufstellung der zu einer Bezugskonfiguration gehörenden Konfigurationseinheiten
- Aufstellung der Historie einer Konfigurationseinheit
- gegenwärtiger Konfigurationsstand
- Statusberichte über Änderungen und Sonderfreigaben
- Aufstellung aller Änderungen seit der letzten Bezugskonfiguration
- Fehlerhäufigkeit pro Modul
- Fehlerursachenreport (klassifiziert)

Berichte dieser Art dienen dazu, den Entwicklungsprozeß auch für Dritte nachvollziehbar darzustellen und Schwachstellen in der Aufbau bzw. Ablauforganisation aufzudecken und korrigierend einzugreifen. Das CCB beispielsweise, benötigt Berichte über den Änderungsstatus der Konfigurationseinheiten, um die Releaseplanung durchzuführen und Sonderfreigaben zu genehmigen oder zurückzustellen.

2.3.4 Konfigurationsaudit

Um sicherzustellen, daß das Produkt seinen vertraglich spezifizierten Anforderungen entspricht und daß das Produkt in seinen Konfigurationsdokumenten richtig wiedergegeben ist, sollte vor der Annahme einer Bezugskonfiguration, ein Konfigurationsaudit durchgeführt werden. Diese Konfigurationsaudits sollten nach dokumentierten und vereinbarten Verfahren, die auch die benötigen Methoden zur Aufzeichnung und Berichterstattung enthalten, durchgeführt werden. In der Regel gibt es zwei Arten von Konfigurationsaudits:

- Das funktionsbezogene Konfigurationsaudit beeinhaltet die förmliche Überprüfung einer Konfigurationseinheit, ob sie die in ihren Konfigurationsdokumenten festgelegten Leistungen und funktionalen Merkmale erreicht hat. Dazu werden die Anforderungen an Funktion und Leistung der Konfigurationseinheit, wie sie in ihrer funktionellen Bezugkonfiguration ausgedrückt sind, festgestellt, um dann durch Überprüfung der Ergebnisse aus Reviews, Prüfungen und Versuchen zu bestätigen, daß die Anforderungen erfüllt werden.

- Das physische Konfigurationsaudit prüft die Übereinstimmung der „Ist-Konfiguration“ einer Konfigurationseinheit mit seinen Konfigurationsdokumenten. Dazu wird das tatsächlich realisierte und geprüfte Produkt mit seinen Konfigurationsdokumenten verglichen, um eine Übereinstimmung sicherzustellen.

Diese Audits werden in der Regel nur einmal für jede Konfiguration durchgeführt und können auch schrittweise durchgeführt werden. Die an dem Auditierungsverfahren zu beteiligenden Entscheidungsgremien und Fachbereiche müssen festgelegt werden. Die anzuwendenden Auditierungsverfahren sollten festgehalten werden und sich in einem entsprechenden Formblatt für die Auditierungsberichte niederschlagen.

2.4 Hilfsmittel und Werkzeuge des Software-Konfigurationsmanagement

Geeignete Hilfmittel und Werkzeuge sind eine wichtige Voraussetzung für die Durchführung des Software-Konfigurationsmanagements. Ohne sie ist ein sinnvolles Konfigurationsmanagement nur schwer durchführbar. Mit ihnen werden nicht nur Programmcode, Objektdateien und Binärcode, sondern auch Spezifikationen und Entwurfsdokumente, Testdaten und Testprozeduren, sowie Fehlermeldungen und Änderungsanforderungen verwaltet. Folgende Werkzeuge und Hilfsmittel werden im folgenden näher behandelt:

- Software-Konfigurationsmanagementplan (SKM-Plan)
- Projektdokumentationssystem
- Projektbibliothek
- Werkzeuge zur Neukonfiguration von Softwaresystemen
- Werkzeuge zur Versionskontrolle und -verwaltung
- Fehler- Änderungsverfolgungssysteme

Der SKM-Plan regelt den Wirkungsbereich des SKM, die Projektorganisation unter Bezugnahme auf das SKM, die Auswahl geeigneter Verfahren und Werkzeuge, Konventionen zur Unterstützung des SKM und die Art der Zusammenarbeit mit Zulieferern. Mit einem SKM-Plan läßt sich somit die Einführung des SKM in einem Software-Projekt regeln. Der SKM-Plan sollte folgende Punkte beinhalten [IEEE 88]:

- Einführung
    - Zweck

    - Geltungsbereich
    - Definitionen
    - Quellenangaben
- Management
    - Organisation
    - Verantwortlichkeiten
    - Schnittstellen
    - Umsetzung des SKM-Plans
    - Anwendbare Grundsätze, Weisungen und Verfahren
- Aufgaben des SKM
    - Konfigurationsidentifizierung
    - Konfigurationsüberwachung
    - Konfigurationsbuchführung
    - Konfigurationsaudit
    - Werkzeuge, Verfahren und Methoden
    - Kontrolle der Lieferanten
    - Aufzeichnungs-, Archivierungs- und Dokumentationsgrundsätze

Ein wichtige Rolle beim SKM spielt das Projektdokumentationssystem. In jedem Projekt entsteht eine Vielzahl von Dokumenten. Es handelt sich dabei nicht nur um Entwicklungsergebnisse, sondern auch um Berichte, Verträge und Pläne. Die Erreichung der Projektziele hängt auch wesentlich von der ordnungsgemäßen Erstellung, Pflege und Güte dieser Dokumente ab. Im Zusammenhang mit dem Projektdokumentationssystem muß es eine Regelung zum Dokumentenmanagement und zur Dokumentenverwaltung geben. Diese muß im Wesentlichen angeben, wie Dokumente zu klassifizieren sind, wie ihre Identifikation zu erfolgen hat und sie muß Richtlinien für die äußere Form, für die inhaltliche Gliederung, den Verteilerkreis, ihre Ablage und für die Statusübergänge der Dokumente (z.B. in Planung - in Arbeit - in Prüfung freigegeben) enthalten.

Die Basis des SKM kann eine Projektdatenbank bilden. Sie enthält sowohl die Elemente des SKM, als auch sämtliche Elemente, die im Rahmen der Dokumentenverwaltung und

des Dokumentenmanagements identifiziert werden. Sie ist damit ein allgemeines Instrument zur Speicherung und zur Verwaltung von Elementen für alle am Projekt Beteiligten, vom Projektleiter bis zum Entwickler. Die Trennung von Element- und Informationsverwaltung wird als eine wesentliche Eigenschaft der Projektdatenbank angesehen, da die Verwaltung von Elementen und die Auswertung von Informationen an die Funktionalität der Projektdatenbank sehr unterschiedliche Forderungen stellt. Bei der Elementverwaltung geht es darum, Elemente geregelt in die Datenbank einzubringen, aufzubewahren oder auszuleihen. Im Gegensatz zur Elementverwaltung, bei der die eingestellten Elemente lediglich wiedergewonnen werden sollen, muß die Informationsverwaltung die freizügige Auswertung und Verknüpfung der gespeicherten Informationen erlauben.

Werkzeuge zur Neukonfiguration von Software-Systemen helfen beim selektiven Konfigurieren eines Systems in Abhängigkeit von Änderungen. Es werden nur die Teile des Systems neu konfiguriert, beispielsweise kompiliert und gebunden, die sich geändert haben oder die von geänderten Teilen abhängig sind. Um die Eindeutigkeit der Konfigurationselemente vor und nach der Neukonfiguration zu bestimmen, werden meist Zeitmarken verwendet. Die Abhängigkeiten der Software-Elemente untereinander und die nötigen oder gewünschten Bearbeitungen werden durch eine vorgegebene regelbasierte Syntax beschrieben. Neuere Werkzeuge berücksichtigen auch die Belange gemeinsamer Nutzung abgeleiteter Elemente bei paralleler Entwicklung. In diesem Fall reicht es nicht mehr aus sich an den Zeitmarken zu orientieren, um entscheiden zu können, ob eine Komponente erneut übersetzt werden muß. Das System muß dazu exakt wissen, aus welchen Quellelementen ein abgeleitetes Element besteht, um entscheiden zu können, ob ein abgeleitetes Element gemeinsam genutzt werden kann oder ein neues erstellt werden muß.

Werkzeuge zur Versionskontrolle und -verwaltung dienen der ökonomischen Speicherung mehrerer Revisionen und Varianten von Software-Elementen. In einem unter der vollständigen Kontrolle des Werkzeugs stehenden Speicherbereich werden nur die Differenzen zwischen den Versionen gespeichert. Sie bieten einen kontrollierten Zugriff auf die Softwareelemente bei dem keine Änderungen verloren gehen und die ändernde Person identifiziert werden kann. Einige Werkzeuge bieten auch die Möglichkeit die einzelnen Änderungen an den Software-Elementen mit dem Änderungsanlaß in Beziehung zu setzen. Damit lassen sich Aussagen zum Änderungstand treffen sowie Konflikte bei konkurrierenden Änderungen erkennen.

Werkzeuge zur Fehler- und Änderungsverfolgung speichern die Änderungsanträge und Fehlermeldungen in einer Datenbank. Alle Informationen, die sich im Laufe der Zeit zu diesen Anträgen und Meldungen ansammeln, werden in der Datenbank gespeichert. Diese gibt somit einen Überblick über die Änderungshistorie und den Bearbeitungsstatus eines Eintrags. Einige Systeme speichern auch Verbindungen zu Konfigurationselementen im Versionsverwaltungssystem. Damit läßt sich Übersicht darüber gewinnen, welche Konfigurationselemente von z.B. einer Fehlerbehebung betroffen sind. Auch Prozeßsteuerungsmöglichkeiten bieten einige Werkzeuge. Ein vorgegebener Prozeßablauf wie z.B. ein Szenario in dem ein Benutzer einen Fehler meldet, ein Entwickler den Fehler analysiert und eine Lösung findet, ein Gremium diese genehmigt, der Entwickler den Fehler behebt und die Qualitätssicherung die Fehlerbehebung prüft, kann durch ein Werkzeug unterstützt werden, indem das Werkzeug dafür sorgt, daß die richtige Reihenfolge bei den Arbeitsschritten eingehalten wird und nur autorisierte Mitarbeiter die jeweiligen Arbeitsschritte durchführen können. Das Werkzeug kann aber noch weiter gehen und nicht nur Prüfungen durchführen, sondern aktiv den Änderungsprozeß steuern. Dazu werden die für einen Arbeitsschritt zuständigen Personen von dem System informiert. Diese vom Werkzeug initiierte Arbeitsablaufsteuerung wird häufig über e-mail verwirklicht.

2.5 Anforderungen an Software-Konfigurationsmanagementsysteme

Wenn man die Anforderungen der ISO 9000 an das SKM betrachtet, so zeigt sich, daß diese recht allgemeiner Natur sind und somit wenig Hinweise für eine Konkretisierung bzw. Strukturierung bieten. Um die Anforderungen zu strukturieren, ist eine Gliederung in funktionsübergreifende Anforderungen, die für alle Funktionsbereiche von SKM-Systemen gelten, sowie in funktionsspezifische Anforderungen geeignet[Ovum 99]. Im folgenden werden diese Anforderungen näher betrachtet.

2.5.1 Funktionsübergreifende Anforderungen

Die funktionsübergreifenden Anforderungen gliedern sich wie folgt:

- Allgemeine Anforderungen (Matrix 1.1)[17]

[17] Die nachgestellte Numerierung in „(..)“ entspricht der aus der Matrix in Anhang A

- Benutzerfreundlichkeit (Matrix 1.1.1)
  Benutzerfreundlichkeit ist ein wesentlicher Aspekt für die Akzeptanz eines Softwaresystems. Bei einem SKM-System liegt hierbei der Schwerpunkt auf der effektiven und effizienten Unterstützung der unterschiedlichen Teammitglieder. Dies kann z.B. durch intuitive grafische Oberflächen mit mächtigen Verwaltungs- und Automatisierungsfunktionen oder eine harmonische Integration in die Entwicklungsumgebung verwirklicht werden.

- Performance / Skalierbarkeit (Matrix 1.1.2)
  Bei größeren Softwareprojekten mit einer großen Anzahl von KE ist es wichtig, daß die Verwaltung und der Zugriff auf die KE den Softwareentwicklungsprozeß nicht behindert. Das System sollte aufgrund seiner Architektur Möglichkeiten zur Skalierung bieten. Einige Systeme verwenden Client/Server-Architekturen um diese Anforderung erfüllen zu können. Die Performance eines SKM-Systems ist ein wesentlicher Faktor für die Akzeptanz des Systems bei den täglichen Anwendern wie Entwickler und QM-Mitarbeiter[18].

- Zugriffsschutz / Rollenkonzept (Matrix 1.1.3)
  Auf unterschiedlichen Ebenen definierbare Zugriffsrechte sowie ein Rollenkonzept sind wichtige Voraussetzungen für einen sicheren Umgang mit den KE. Dadurch wird gewährleistet, daß KE nicht unbeabsichtigt oder unberechtigt verändert werden. Das Rollenkonzept erleichtert die Verwaltung unterschiedlicher Zugriffsprofile in größeren Teams. Bei Client/Server-Architekturen kommen Funktionen zur Verschlüsselung für den sicheren Zugriff auf zentrale Daten hinzu.

- Datensicherung (Matrix 1.1.4)
  Die regelmäßige Datensicherung ist eine notwendige Bedingung, um im Falle von Systemausfällen die Entwicklungsarbeit zeitnah und ohne größere Verluste an Entwicklungsergebnissen wiederaufnehmen zu können. Die Daten in einem SKM-System sind für eine softwareproduzierende Organisation ebenso wichtig wie z.B. die Versicherungsvertragsdaten für ein Versicherungsunternehmen.

- Plattformunabhängigkeit (Matrix 1.1.5)
  Die Verfügbarkeit auf unterschiedlichen Plattformen ist ein zunehmend wichtiger Faktor, da in Unternehmen häufig unterschiedliche Plattformen anzutreffen sind und somit beispielsweise der Client unter Windows NT und

[18] Mitarbeiter im Bereich Qualitätsmanagement

der Server unter Unix implementiert ist. Eine verschiedene Plattformen integrierende Umgebung, verhindert somit redundante und inkonsistente Daten im SKM-System.

- Anpaßbarkeit / Erweiterbarkeit (Matrix 1.1.6)
  Da jedes Entwicklungsprojekt seine spezifischen Anforderungen an die SKM-Umgebung hat, sind Änderungs- und Erweiterungsmöglichkeiten eine wichtige Voraussetzung, um die SKM-Umgebung in den gesamten Entwicklungsprozeß individuell zu integrieren. Dazu gehören z.B. die Integration mit Testwerkzeugen und Projektmanagementsystemen.
- Flexible konfigurierbare Reports (Matrix 1.1.7)
  Die vielfältigen Daten innerhalb eines SKM-Systems, dazu gehören z.B. Daten über die KE, Änderungsmitteilungen sowie Metainformationen, eröffnen prinzipiell eine Vielzahl von Auswertungsmöglichkeiten. Daher sind individuell konfigurierbare Reports ein wichtiges Element zur Überwachung, Messung und Verbesserung des Entwicklungsprozesses.

- Team-Unterstützung (Matrix 1.2)
  - Verteilte Softwareentwicklung (Matrix 1.2.1)
    Große internationale Projekte oder aber auch flexible Heimarbeitsplätze bedingen geographisch verteilte Entwicklungsteams. Diese voneinander getrennten Teams sind aufgrund der nur indirekt möglichen Kommunikation auf qualifizierte Informationen zum Entwicklungsstand des Projektes und der KE angewiesen. Auf der anderen Seite müssen die unterschiedlichen Entwicklungsergebnisse der Teams komfortabel in ein zentrales SKM-System integriert werden können.
  - Verteilte Datenhaltung (Matrix 1.2.2)
    Um verteilte Entwicklung effizient ermöglichen zu können, kann eine lokale, also auf Entwicklungsstandort bezogene, Verwaltung von Teilen der KE sinnvoll sein. Diese Anforderung kann über Repliziermechanismen, wie sie aus der Datenbankwelt bekannt sind, verwirklicht werden.
  - Unterstützung von Web-Technologie (Matrix 1.2.3)
    Ein Zugriff über Standard Browsertechnologie ermöglicht eine einfache Anbindung bzw. Integration der Kunden in das SKM-System. Es können somit sowohl Änderungs- und Fehlermeldungen übermittelt werden, als auch

Statusabfragen zu Meldungen ohne Installationsaufwand auf der Benutzerseite realisiert werden. Aber auch für geographisch verteilte Teams bietet sich hiermit eine kostengünstige Anbindungsmöglichkeit.

- Prozeßunterstützung (Matrix 1.3)
  - Flexible Modellierung der Entwicklungsprozesse (Matrix 1.3.1)
    Das flexible Modellieren des Entwicklungsprozesses beinhaltet die Definition von Abhängigkeiten und Regeln für die Behandlung der unterschiedlichen KE. Somit können bestimmte Reihenfolgen bzw. Vor- und Nachbedingungen für die entsprechenden Entwicklungsschritte konfiguriert werden. Dies schafft auf der einen Seite eine höhere Prozeßtransparenz, kann jedoch auf der anderen Seite die Teammitglieder in ihrem Entfaltungsfreiraum einschränken.
  - Steuerung und Kontrolle der Entwicklungsprozesse (Matrix 1.3.2)
    Die Steuerungs- und Kontrollfunktionen gewährleisten die Einhaltung der definierten Regeln und unterstützen aktiv den täglichen Arbeitsablauf der Teammitglieder durch Weiterleitungs- bzw. Benachrichtigungsfunktionen
  - Automatisierung von Prozessen (Matrix 1.3.3)
    Einige Routineaufgaben bzw. Massenverarbeitung lassen sich mit einer grafischen Oberfläche häufig nur unzureichend realisieren. Ein Command-Line Interface bzw. Programmierschnittstellen bieten hierfür eine Möglichkeit zur Automatisierung.

- Integration (Matrix 1.4)
  - Integration mit Entwicklungswerkzeugen (Matrix 1.4.1)
    Die Integration mit den Entwicklungswerkzeugen ist eine wichtige Anforderung für die reibungsarme Einbindung von SKM-Funktionen in den Entwicklungsprozeß. Insofern sollten die SKM-Funktionen die eigentlichen Entwicklungsarbeiten nicht behindern oder erschweren.
  - Integration mit Testwerkzeugen (Matrix 1.4.2)
    Eine Verbindung von SKM- und Testwerkzeugen ermöglicht eine integrierte Änderungs-/Fehlerverwaltung mit weniger Redundanz durch z.B. automatisch generierte Fehlermeldungen als Ergebnis von Testfällen. Darüber hinaus ist eine automatische Erkennung von durchzuführenden Tests bei gegebenen Änderungen denkbar.

- Integration mit Projektmanagementwerkzeugen (Matrix 1.4.3)
  Im SKM-System werden Informationen zum Status von einzelnen Aufgaben im Projekt verwaltet. Eine Verbindung zu einem Projektmanagementsystem ermöglicht die einfache Ergänzung dieser Informationsbasis um Personaleinsatz- Termin- und Kostenplanung.

2.5.2 Funktionsspezifische Anforderungen

- Versionsmanagement (Matrix 2.1)
  - Sicheres Ein- und Auschecken der KE (Matrix 2.1.1)
    Ist Voraussetzung für paralleles Arbeiten. Es wird ein gesicherter einheitlicher Zugriff auf die KE gewährleistet, der gegenseitiges Überschreiben der Entwicklungsergebnisse mehrerer Entwickler verhindert.
  - Unterstützung von Promotionsstufen (Matrix 2.1.2)
    Mit Hilfe von Promotionsstufen kann der unterschiedliche Reifegrad eines KE während des Entwicklungsprozesses abgebildet werden. Denkbare Promotionsstufen sind z.B. „In Entwicklung“, „Test“ und „Produktion“.
  - Prozeßunterstützung (Matrix 2.1.3)
    Die Prozeßunterstützung im Rahmen des Versionsmanagement orientiert sich an den wesentlichen Funktionen wie z.B. Einchecken, Auschecken oder einem Wechsel der Promotionsstufe. Mit diesen Ereignissen können individuell konfigurierbare Aktionen verknüpft werden, die den gewünschten Prozeß unterstützen.
  - Umgebungsmanagement (Matrix 2.1.4)
    Das Umgebungsmanagement sorgt für einen einfachen und komfortablen Zugriff der Entwickler oder QM-Mitarbeiter auf die unterschiedlichen Releasestände eines Softwaresystems. Das bedeutet, daß z.B. ohne erheblichen Aufwand ein älterer beim Kunden ausgelieferter Stand einem Entwickler zur Korrektur eines Fehlers bereitgestellt werden kann, ohne daß andere Entwickler oder die Hauptentwicklungslinie davon beeinflußt werden.
  - Unterstützung mehrerer Entwicklungslinien (Matrix 2.1.5)
    Mehrere Entwicklungslinien sind für die Verwaltung von mehreren Releaseständen und Varianten einer Software notwendig. Zur Zusammenführung unterschiedlicher Entwicklungslinien sollten leistungsfähige

grafische Werkzeuge zur Verfügung stehen, die auch teilweise automatisch Änderungen unterschiedlicher Entwickler an der selben KE integrieren.

- Verwaltung unterschiedlichster KE (Matrix 2.1.6)
  Das Versionsverwaltungssystem muß in der Lage sein unterschiedliche Dateitypen effizient verwalten zu können. Dazu gehören ASCII-Dateien wie z.B. Quellcode, Binärdateien wie Objektcode oder ausführbare Programme aber auch Datenbankstände die im Binärformat vorliegen.
- Komfortable Abbildung der Produktstruktur (Matrix 2.1.7)
  Die Produktstruktur sollte z.B. in Form der Verzeichnisstruktur eines Projekts in dem Versionsmanagement abbildbar sein. Ist dies nicht möglich, müssen sich die Teammitglieder an eine für das Versionsmanagement eigene Struktur gewöhnen, die von der Struktur der Entwicklungsumgebung abweicht.
- Integration mit dem Änderungsmanagement (Matrix 2.1.8)
  Die Integration mit dem Änderungsmanagement ist ein integraler Bestandteil für die Durchgängigkeit eines SKM-Systems. Sie gewährleistet die Zuordnung von Änderungs- oder Fehlermeldungen zu Revisionen bzw. Varianten von betroffenen KE. Von der Qualität dieser Integration hängen z.B. zeitnahe Projektstatusreports ab.

- Änderungsmanagement (Matrix 2.2)
  - Flexibel gestaltbare Änderungs- bzw. Fehlermeldungen (Matrix 2.2.1)
    Dieser Punkt beinhaltet z.B. die Definition von individuellen Feldern auf den Meldungsformularen sowie die Hinterlegung spezifischer Prüflogik. Darüber hinaus sollte ein flexibles Layout-System unterschiedliche Sichten auf das Änderungsmanagement unterstützen.
  - Integration mit dem Versionsmanagement (Matrix 2.2.2)
    Es gilt hier das Gleiche wie für den analogen Punkt unter Versionsmanagement.
  - Prozeßunterstützung (Matrix 2.2.3)
    Bei diesem Punkt sind Funktionen wie die Verwaltung individueller to-do-Listen oder die automatische Benachrichtigung bestimmter Teammitglieder über die Änderungen eines anderen Teammitgliedes gemeint. Zusätzlich sollten Abhängigkeiten wie z.B. „Änderung an KE darf nur erfolgen, wenn eine Änderungs- oder Fehlermeldung assoziiert wurde“ definierbar sein.
  - Trend- und Analysereports (Matrix 2.2.4)
    Darunter sind z.B. Reports zur Analyse von Fehlerhäufigkeit pro Modul,

durchschnittliche Fehlerkorrekturdauer gewichtet nach Schwere des Fehlers oder die durchschnittliche Anzahl offener Änderungs- und Fehlermeldungen zu verstehen.

- Releasenotes (Matrix 2.2.5)
  Die automatische Verwaltung und Generierung der Releasenotes aus dem Änderungsmanagement gewährleistet eine einheitliche und vollständige Dokumentation der Änderungen ein einem System von Release zu Release für den Kunden.
- Statusreports (Matrix 2.2.6)
  Statusreports zu bestimmten KE helfen z.B. dem Kundensupport bei der Beantwortung von Fragen zum Status einer Meldung oder bei der Überprüfung ob ähnliche Meldungen bereits erfaßt und in Arbeit sind.
- Einbindung beliebiger Dokumenttypen (Matrix 2.2.7)
  Die Einbindung von Standard-Dokumenttypen wie z.B. Word-Dokumenten oder Grafik-Hardcopys, ist ein wichtiger Faktor um Fehlermeldungen des Kunden mit allen notwendigen Informationen im System einfach verwalten zu können.

- Buildmanagement (Matrix 2.3)
  - Regelbasierte Verwaltung/Steuerung der Abhängigkeiten von KE (Matrix 2.3.1)
    Dieser Punkt beinhaltet die Definition von Ableitungsregeln, die festlegen, in welchen Schritten und Reihenfolgen welche abgeleiteten Typen von KE aus welchen Quell KE erstellt werden. Diese Regeln sind Voraussetzung für eine vollständig automatische Systemgenerierung.
  - Integration mit dem Versionsmanagement (Matrix 2.3.2)
    Damit auch das Buildmanagement den einheitlichen und sicheren Zugriffsweg auf die KE verwenden kann, muß eine Integration mit dem Versionsmanagement vorhanden sein, die einen performanten Zugriff auf die im Versionsmanagement verwalteten KE gewährleistet.
  - Unterstützung des Auslieferungsverfahrens (Matrix 2.3.3)
    Das Buildmanagement sollte in der Lage sein, sowohl komplette Releases eines Softwareprodukts als auch selektive Korrekturen oder Ergänzungen zu erzeugen. Die Verwaltung der Auslieferungspakete und eine Zuordnung zu entsprechenden Kunden ist ebenfalls wünschenswert.

- Unterstützung von Footprinting (Matrix 2.3.4)
  Die Footprinting-Funktion gewährleistet eine lückenlose Rückverfolgung der Bestandteile einer auslieferbaren KE. Dazu wird in den abgeleiteten KE die vollständige Liste der sie erzeugenden Quell-KE mit Revisions- und Variantenbezeichnung abgelegt. Hierdurch wird der Systemgenerierungsprozeß transparent.

2.5.3 Sichtenspezifische Anforderungen

Eine Bewertung der Anforderungen aus unterschiedlichen Sichten trägt der Tatsache Rechnung, daß die Benutzer eines SKM-Systems abhängig von der Rolle in der sie agieren, verschiedene Schwerpunkte setzen. Zur Ermittlung der sichtenspezifischen Anforderungen wurden die Mitglieder des PRORIS-Projektteams sowie einige PRORIS-Koordinatoren der Kunden befragt. Dabei wurden vier unterschiedliche Rollen bzw. Sichten identifiziert:

- Kundensicht
- Projektmanagementsicht
- Qualitätsmanagementsicht
- Entwicklersicht

Die Kundensicht wurde von den PRORIS-Koordinatoren der Kunden eingenommen. Zu ihrer Aufgabe gehört es als kompetenter Ansprechpartner für die Mitarbeiter der Fachabteilungen beim Kunden und der CTH zu fungieren. Die PRORIS-Koordinatoren kennen das PRORIS-System und die spezifischen Probleme der jeweiligen Mitarbeiter beim Kunden sehr gut und wirken somit als Filter, der für eine effiziente Kommunikation zwischen dem Kunden und der CTH sorgt. Insofern liegt ihr Schwerpunkt auf einer einfachen und schnellen Lösung für Problemmeldungen sowie stets aktuellen Statusreports.

Ein PRORIS-Projektmanager leitet ein PRORIS-Teilprojekt bzw. eine spezifische PRORIS-Einführung bei einem Kunden. Er ist in diesem Zusammenhang mit typischen Projektmanagementfragestellungen bezüglich Zeit, Kosten und Personalressourcen konfrontiert. Daraus ergibt sich ein Schwergewicht im Bereich Steuerung und Kontrolle des Entwicklungsprozesses.

Die Sicht des Qualitätsmanagements wird von den SKM-Beauftragten sowie den jeweiligen Softwaretestern eingenommen. Die SKM-Beauftragten managen die Entwicklungsumgebung und sorgen für einen geregelten Ablauf von Entwicklung über Test bis zum nächsten Release.

Die Sicht der Entwickler ist durch ihre tägliche Entwicklungsarbeit geprägt. Ihnen geht es in diesem Zusammenhang darum mit möglichst wenig zusätzlichen Aufwand durch die Verwendung des SKM-Systems belastet zu sein. Auf der anderen Seite brauchen sie leistungsfähige Werkzeuge Rückverfolgung und Fehleranalyse.

Die Befragung wurde mit Hilfe eines Fragebogens durchgeführt, der die Wichtigkeit der unter 2.5.1 und 2.5.2 beschriebenen Anforderungen aus Sicht der jeweiligen Rolle aufnimmt. Die Wichtigkeit konnte mit folgenden Werten ausgedrückt werden:

- 0 für irrelevant
- 1 für wünschenswert
- 2 für wichtig
- 3 für notwendig

Über die einzelnen Ergebnisse pro Sicht und Anforderung wurde der Mittelwert gebildet und anschließend aufgerundet. Der Autor hat an Stellen, an den das Ergebnis der Befragung von seiner Auffassung abweicht, entsprechend seinen Sollwert in „(...)“ hinter dem Umfrageergebnis vermerkt.

Die befragten PRORIS-Koordinatoren konnten per Definition nur aus der Kundensicht ihre Bewertung abgeben. Den Personen im PRORIS-Projektteam wurde freigestellt aus welcher Sicht sie den Fragebogen ausfüllen. Es konnten auch mehrere Sichten von einer Person ausgefüllt werden. Dies liegt darin begründet, daß von ein und derselben Person unterschiedliche Rollen ausgeübt werden und somit unterschiedliche Sichten auch von einer Person vertreten werden können. Diese Art der Befragung hat den Projektteilnehmern deutlich aufgezeigt, daß abhängig von der Rolle in der sie jeweils agieren, sie auch unterschiedliche Schwerpunkte in SKM-Systemen setzen. Damit wird das Verständnis für die Aufgaben der jeweils anderen Rolle gestärkt.

Sicht der Kunden

Aus der Anforderungsmatrix[19] geht hervor, daß aus Kundensicht zwei Schwerpunktbereiche wichtig erscheinen. Der erste Schwerpunkt liegt im Bereich der allgemeinen Anforderungen hinsichtlich Benutzerfreundlichkeit, Performance, Sicherheit, Datensicherung sowie flexiblem Reporting. Der zweite Schwerpunkt liegt im Bereich Änderungsmanagement, bei dem es dem Kunden um Transparenz hinsichtlich des Status

[19] siehe Anhang A

seiner Fehlermeldungen und Änderungswünsche geht. Der Autor sieht hier bei einigen Punkten eine höhere Relevanz aus Sicht der Kunden.

Die Datensicherung ist aus Sicht des Autors ein notwendiger Punkt, da es aus Gründen des Investitionsschutzes für den Kunden gewährleistet sein muß, daß die bei ihm laufende Version als Quellcode vorliegt. Der Unterstützung des Auslieferungsverfahrens sollte ebenfalls eine höhere Beachtung geschenkt werden, da somit Aufwand, auf Seiten des Kunden, beim Einspielen neuer Versionen eingespart werden kann.

Sicht des Projektmanagements

Die Anforderungen des Projektmanagement verteilen sich relativ gleichmäßig über die Anforderungsmatrix. Dennoch ist ein Schwerpunkt auf der Auswertungsseite zu erkennen, der das Projektmanagement in die Lage versetzt auf Fehlentwicklungen zu reagieren und somit eine stetige Verbesserung des Entwicklungsprozesses zur ermöglichen. Ein weiterer wichtiger Aspekt ist eine Schnittstelle zu einem Projektmanagementsystem, damit der Projektleiter nicht fern der Realität seine Planung vornimmt und den engen Kontakt zum operativen Geschehen behält.

Die Automatisierung von Prozessen sollte, nach Meinung des Autors, aus Sicht des Projektmanagements als notwendig angesehen werden, da somit der Entwicklungsprozeß wesentlich effizienter und transparenter gestaltet werden kann und somit die Entwickler von lästigen manuellen Qualitätssicherungsaufgaben entlastet werden können. Dies ist eine wichtige Voraussetzung dafür, daß die Entwickler das SKM-System annehmen.

Sicht des Qualitätsmanagements

Für das Qualitätsmanagement sind fast alle Anforderungen der Anforderungsmatrix wichtig. Dieser Umstand ist auch nicht weiter verwunderlich, da es sich bei einem SKM-System um ein Kernsystem zur Unterstützung des Qualitätsmanagement handelt.

Die Automatisierung von Prozessen sollte, nach Meinung des Autors, gerade auch aus Sicht des Qualitätsmanagements als notwendig angesehen werden[20].

Sicht der Anwendungsentwickler

[20] siehe „Sicht des Projektmanagements“

Aus Sicht der Entwickler liegen ein Schwerpunkte auf Performance, Benutzerfreundlichkeit sowie der Integration in die bestehende Entwicklungsumgebung. Werden diese Anforderungen nicht befriedigend erfüllt, kommt es zu unnötigen Verzögerungen bei den täglichen Entwicklungsarbeiten. Das führt im schlimmsten Fall zur Umgehung der Regeln im SKM-System und somit zu einer nicht mehr aussagekräftigen Datenbasis. Ein weiterer Schwerpunkt liegt bei dem Komfort und der Funktionalität des Versionsmanagement, daß für den Entwickler das Tor zu den KE und damit zu dem Programmcode darstellt.

Anmerkungen des Autors zur Befragung

Es muß an dieser Stelle darauf hingewiesen werden, daß die Befragung in mehrerlei Hinsicht nicht als repräsentativ anzusehen ist. Es wurden nur relativ wenig Personen befragt, die teilweise mehrere Rollen ausüben. Die Rollen sind theoretisch klar voneinander zu trennen. Die befragten Personen fällt eine solche strikte Trennung natürlich schwer. Insofern können die Ergebnisse, durch solche Doppelfunktionen bestimmter Projektteilnehmer beeinflußt sein. Dennoch sind, wie oben aufgezeigt, einige grundlegende Unterschiede in der Priorisierung von SKM-Anforderungen aus Sicht unterschiedlicher Rollen erkennbar.

3 Aufbau der PRORIS Entwicklungsumgebung

In diesem Kapitel wird die Entwicklungsumgebung von PRORIS dargestellt. Der Schwerpunkt hierbei liegt auf der eingesetzten Hard- und Software, der Integration in die Netzwerkumgebung, der Performance sowie der Datensicherung und Plattformunabhängigkeit der Umgebung.

Die Entwicklung von PRORIS begann ursprünglich ausschließlich unter OS/2 und sollte auch nur unter OS/2 als Zielplattform zum Einsatz kommen. Die Entwicklungen der letzten Jahre auf dem Betriebssystemmarkt erforderten es jedoch 1997 PRORIS auf Windows NT zu migrieren. Da aber OS/2 bei vielen Kunden noch im Einsatz war und teilweise auch noch ist, muß die Entwicklungsumgebung, und selbstverständlich auch PRORIS, beide Betriebssystemplattformen unterstützen können. Zunächst wird auf die Serverseite der Entwicklungsumgebung näher eingegangen.

3.1 Server in der Entwicklungsumgebung

Um einen weitgehend ausfallfreien Betrieb zu ermöglichen, kommt bewährte Serverhardware zum Einsatz. Dazu gehören z.B. redundante Plattensysteme und fehlertoleranter Speicher. Alle Systeme sind mit einer unterbrechungsfreien Stromversorgung ausgerüstet und in einem speziellen Serverraum aufgestellt. Dieser Raum wird permanent auf ca. 19° Celsius gekühlt, um die Maschinen zu schonen. Der Zugang zu diesem Raum ist nur befugtem Mitarbeitern gestattet um absichtliche oder versehentliche Beschädigungen an Hard- und Software zu verhindern.

Die Entwicklungsserver teilen sich in OS/2- und Windows NT-Serversysteme auf. Ihre Aufgaben ergeben sich aus dem Umstand, daß PRORIS sowohl unter OS/2 als auch unter Windows NT weiterentwickelt und gewartet werden muß. Die OS/2 und die Windows NT Umgebung wird jeweils in einer eigenen Domäne verwaltet. Unter einer Domäne versteht man eine Verwaltungseinheit, die einer Gruppe von Benutzern Ressourcen, wie z.B. Plattenplatz oder Drucker, auf einem oder mehreren Servern zur Verfügung stellt.

3.1.1 OS/2 Server

Domänen-Controller / Dateiserver

Zu der OS/2-Domäne gehören vier Server. Ein Server ist der sogenannte Domänen-Controller, der eine besondere Rolle unter den Servern einnimmt und für die Verwaltung der Domäne verantwortlich ist. Zusätzlich stellt dieser Server alle

Entwicklungslaufwerke[21] für die PRORIS-Entwickler sowie sämtliche Kundenumgebungen[22] zur Verfügung. Der Server läuft unter dem Betriebssystem OS/2 Warp 4.0 mit dem sogenannten Fixpack 10, welches die "Jahr-2000"-Korrekturen beinhaltet. Als Netzwerkbetriebsystem kommt der LAN Server 5.0 von IBM zum Einsatz.

Datenbankserver

Die restlichen drei Server in der OS/2-Domäne sind Datenbankserver. Es müssen drei Datenbankserver sein, da bei den PRORIS-Kunden unterschiedliche Versionen des Datenbanksystems DB2, oder wie inzwischen umbenannt UDB (Universal Database), zum Einsatz kommen. Da sich diese Versionen vom Verhalten her teilweise unterscheiden, sind für eine adäquate Unterstützung der Kunden bei Wartungsanfragen, alle bei Kunden im Einsatz befindlichen Versionen vorzuhalten. Das sind z.Zt. unter OS/2 drei Versionen:

- DB2/2 1.2 mit Fixpack 7047
- DB2 für OS/2 2.1.2 mit Fixpack 8090
- UDB 5.2 für OS/2

Die Datenbanken auf den Servern werden täglich in das Dateisystem gesichert. Einer der Datenbankserver verfügt über ein entsprechendes Sicherungslaufwerk, über das täglich alle OS/2 und NT-Server gesichert werden. Die vorherige Sicherung der Datenbanken in das Dateisystem ist notwendig, da momentan nur so eine konsistente Sicherung der Datenbanken gesichert werden kann. Auf allen OS/2 Datenbankservern kommt, wie beim Domänen-Controller, OS/2 Warp 4 mit Fixpack 10 sowie LAN Server 5.0 zum Einsatz.

3.1.2 Windows NT Server

Domänen-Controller / Dateiserver

Zu der Windows NT-Domäne gehören zwei Server. Ein Server ist, wie in der OS/2 Umgebung der Domänen-Controller. Auch unter Windows NT hat der Domänen-Controller zusätzlich die Aufgabe Netzwerklaufwerke für das PRORIS-Entwicklungsteam zur Verfügung zu stellen. Dazu gehören Laufwerke mit allen Entwicklungswerkzeugen, Laufwerke für den Datenaustausch im PRORIS-Team oder Verzeichnisse mit weiteren

[21] siehe 3.3
[22] siehe 3.3

Hilfsprogrammen[23]. Zusätzlich stellt dieser Server eine Verbindung zu dem firmenweiten Novell-Netzwerk her. Damit können auch Dateien und Drucker außerhalb des PRORIS-Entwicklungsteams genutzt werden.

Der Server läuft unter Windows NT 4.0 mit dem weitgehend "Jahr-2000"-fähigen Service Pack 5.

Datenbankserver

Auf dem Datenbankserver unter Windows NT kommt lediglich die UDB-Version 5.2 zum Einsatz. Auch hier werden die Datenbanken täglich in das Dateisystem gesichert und anschließend auf das Bandlaufwerk des einen OS/2-Servers gesichert.

Der Server läuft unter Windows NT 4.0 mit dem weitgehend "Jahr-2000"-fähigem Service Pack 5.

Weitere Server

Über die beiden genannten Domänen hinaus gibt es zwei weitere Server, die für die PRORIS-Entwicklungsumgebung von Bedeutung sind. Zum einen handelt es sich um einen Sybase-Datenbankserver Version 11.9.2, auf dem die Meldungsdatenbank[24] für PRORIS verwaltet wird. Zum anderen um einen Windows Terminal Server der PRORIS unter Windows NT einen Mehrbenutzerbetrieb erlaubt.

Für die Meldungsdatenbank kann kein UDB-Server eingesetzt werden, da das Werkzeug zur Unterstützung des Änderungsmanagements (PVCS Tracker), UDB als Datenbanksystem nicht unterstützt.

Der Einsatz von PRORIS auf dem Windows Terminal Server schafft die Möglichkeit, PRORIS auch über WAN-Strecken zugänglich zu machen. Hierbei laufen die PRORIS-Prozesse und ggf. auch die gesamte Windows-Oberfläche zentral auf einem Terminal Server ab. Auf der Arbeitsstation wird lediglich die Oberfläche angezeigt. Da nur die Veränderungen an der Oberfläche über die Leitung geschickt wird, ist es möglich auch über Verbindungen ab 19Kbit/s PRORIS zu betreiben.

[23] siehe 3.3
[24] siehe Abschnitt 4.2.2

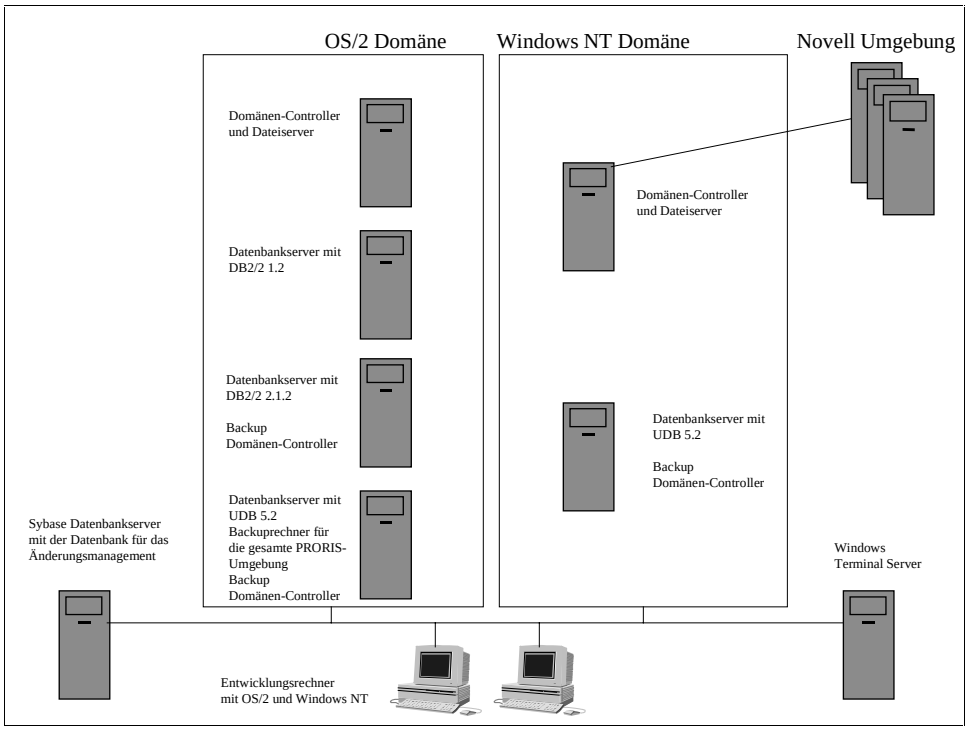


Abb. 3-1: PRORIS Entwicklungsumgebung

3.2 Standard Entwicklungsrechner

Auf den Entwicklungsrechnern muß sowohl unter OS/2 als auch unter Windows NT entwickelt werden können. Dazu wird die Festplatte in mehrere Partitionen aufgeteilt. Über einen sogenannten Boot-Manager kann dann entweder OS/2 oder Windows NT gebootet werden.

Bei einigen Kunden kommen ältere Datenbankversion als UDB 5.2 zum Einsatz. Daher weichen diese Rechner von der weiter unten beschriebenen Standardkonfiguration ab. Auf einem Rechner kommt DB2/2 in der Version 1.2 und auf einem anderen DB2 für OS/2 Version 2.1.2 zum Einsatz. Diese Rechner dienen ausschließlich der Erstellung der PRORIS-Versionen für die oben genannten Datenbankversionen.

3.2.1 OS/2 Partition

Auf der OS/2 Partion ist OS/2 Warp 4 mit Fixpack 10 als Betriebssystem installiert. Die Clientsoftware für die PRORIS-Datenbankserver ist UDB Version 5.2. Mit Dieser Version der Clientsoftware ist es möglich auch auf ältere Versionen des Datenbankmanagementsystem serverseitig zuzugreifen.

Als C Compiler und Debugger kommt der IBM C Set 2.1 mit den entsprechenden Betriebssystembibliotheken zum Einsatz. Dieser Compiler hat sich trotzt seines Alters als sehr zuverlässig erwiesen. Der Compiler und die Bibliotheken werden zentral auf Serverlaufwerken verwaltet und können somit nicht versehentlich manipuliert werden.

Für das Versionsmanagement wird PVCS Version Manager 6.0 in Form einer Netzwerkinstallation verwendet. Das heißt, daß der überwiegende Teil der Installation auf einem Serverlaufwerk liegt und somit Plattenplatz auf dem Arbeitsplatzrechner spart. Darüber hinaus gestaltet sich Releasewechsel der Versionverwaltungssoftware einfacher und Releasekonflikte durch unterschiedliche Installationen auf den Entwicklungsrechnern werden vermieden.

Das Buildmanagement wird vom PVCS Configuration Builder 5.3 unterstützt[25]. Hinsichtlich der zentralen Installation gilt für den Configuration Builder das gleiche wie für den Version Manager.

Als Editor bzw. IDE (Integrated Development Enviroment) kommt die PWB (Programmer's Workbench) von Microsoft zum Einsatz. Die Workbench konnte, sowohl unter OS/2 als auch unter Windows NT, gut mit dem PVCS Configuration Builder integriert werden. Auch die Workbench wird zentral auf Serverlaufwerken verwaltet.

Der Zugriff auf die Änderungsdatenbank kann nur über die in OS/2 integrierte Windows 3.1 Funktionalität ermöglicht werden, da die entsprechende Software PVCS Tracker 5.0 nur in einer Windows- und keiner OS/2-Version vorliegt. Damit auf den zuvor erwähnten Sybase Datenbankserver zugegriffen werden kann, ist auf dem Windows 3.1 Teil der OS/2 Installation der Sybase-Windows-Client 11.9.2 installiert.

Darüber hinaus gehört zur Standardinstallation der Netscape Browser 4.04 für OS/2.

Der Zugriff auf Novell-Ressourcen erfolgt über den zuvor erwähnten Windows NT Domänen-Controller, der Novell-Laufwerke und Novell-Drucker zur Verfügung stellt.

3.2.2 Windows NT Partition

Auf der Windows NT Partition ist Windows NT 4.0 mit Fixpack 5 und dem Internet Explorer 5 installiert. Als Office Umgebung kommt Microsoft Office 97 zum Einsatz. Wie unter OS/2 wird auch die Clientsoftware von UDB 5.2 für den PRORIS-Datenbankzugriff verwendet.

Als C Compiler und Debugger wird Microsoft Visual C++ 5.0 eingesetzt. Dazu gehören auch Werkzeuge für die Erstellung von Dialogen und Hilfedateien. Der Compiler ist

[25] siehe Abschnitt 5.3

wiederum auf Serverlaufwerken installiert, während der Debugger lokal installiert werden muß.

Als Editor und IDE kommt wie zuvor erwähnt die PWB von Microsoft zum Einsatz.

Für die SKM-Produkte, PVCS Version Manager, PVCS Configuration Builder und PVCS Tracker gilt das Gleiche wie unter 3.2.1. PVCS Tracker liegt zur Zeit in der Version 6.5 vor. Diese Version wurde erhebliche Verbesserungen für das Änderungsmanagement beinhalten. Sie kann jedoch nicht eingesetzt werden, da PVCS Tracker Version 5, die höchste von Windows 3.1 unterstützte Version ist. Solange OS/2 mit integriertem Windows 3.1 für PRORIS noch eine Entwicklungsplattform darstellt, können diese Verbesserungen nicht eingeführt werden.

Unter Windows NT erfolgt der Zugriff auf Novell-Ressourcen direkt über den Novell-Client für Windows NT.

3.3 Umgebungsmanagement

Da PRORIS ursprünglich ausschließlich für OS/2 entwickelt wurde, ist OS/2 auch nach der Migration auf Windows NT immer noch die führende Entwicklungsplattform für PRORIS. Dieser Zustand wird noch solange anhalten, bis der letzte Kunde OS/2 als Client-Betriebssystem abgelöst hat. OS/2 muß für diese Zeit die führende Entwicklungsplattform bleiben, da die laufende automatische Migration von z.B. Dialogdefinitionen oder Hilfedateien, nur von OS/2 in Richtung Windows NT funktioniert[26]. Dieser Umstand sorgt dafür, daß neue Entwicklungen die unter der Windows NT Plattform genutzt werden könnten nicht implementiert werden bis eine vollständige Ablösung von OS/2 durchgeführt wird.

Die PRORIS-Umgebungen unterteilen sich in Entwicklungsumgebung und Kundenumgebungen. In der Entwicklungsumgebung wird an den jeweils aktuellen Revisionen der PRORIS-Dateien gearbeitet um neue Anforderungen, Änderungswünsche oder Fehlerkorrekturen niedriger Dringlichkeit umzusetzen. Darüber hinaus ist für jeden Kunden eine entsprechende Kundenumgebung eingerichtet, in dem das letzte beim Kunden ausgelieferte Release verwaltet wird. Bei den Kunden wiederum ist eine Test- und eine Produktionsumgebung eingerichtet. Aus Sicht eines Kunden, wird somit ein neues Release zuerst in der jeweiligen Kundenumgebung bei der CTH erstellt. Nach den entsprechenden Integrationstests und der Freigabe erfolgt die Installation in der Testumgebung des Kunden. Der Kunde testet dann das neue Release in seiner Testumgebung und installiert nach entsprechender Freigabe das neue Release in Produktion. Bei der CTH wird immer solange

[26] siehe auch Abschnitt 5.3

ein Abbild der Produktionsumgebung des Kunden aufbewahrt, bis ein neues Release in Produktion gegangen ist, um Fehler in der Produktionsumgebung des Kunden nachvollziehen zu können. Der Zugriff auf die unterschiedlichen Umgebungen ist über Benutzer- bzw. Gruppenrechte auf die jeweiligen Netzwerkressourcen geregelt. Auf die Entwicklungsumgebung haben alle PRORIS-Projektmitglieder mit ihren jeweiligen Benutzernamen Zugriff. Für die Kundenumgebungen sind Testbenutzer und Administratoren eingerichtet. Die Testbenutzer werden für die Integrationstests bzw. zur Fehlersuche verwendet. Die Administratoren für die Kundenumgebungen führen die Integration- bzw. Korrekturarbeiten durch. Durch die Trennung der Aufgaben auf verschiedene Benutzerrollen, wird das Risiko von ungewollten Löschungen oder Veränderungen in der falschen Umgebung verringert.

Das Verfahren für jeden Kunden eine eigene Umgebung vorzuhalten, hat sich als sehr kundenfreundlich aber auch sehr aufwendig erwiesen. Aus diesem Grund wird in Zukunft lediglich eine aktuelle Integrationsumgebung für alle Kunden vorgehalten. Korrekturen auf einem Release werden somit nicht mehr individuell pro Kunde eingespielt, sondern als allgemeine Korrekturen auf dem Release betrachtet. Das bedeutet, daß ein Kunde, der eine neue Korrektur eingespielt haben möchte, auch alle vorherigen Korrekturen seit dem letzten Release einspielen muß. Somit gibt es jeweils nur einen Entwicklungsstrang pro ausgeliefertem Release. Dieses Vorgehen wird die Wartungsaktivitäten wesentlich vereinfachen und für höhere Qualität der Releases sorgen, da die Kunden nicht mehr individuelle Releasestände einsetzen.

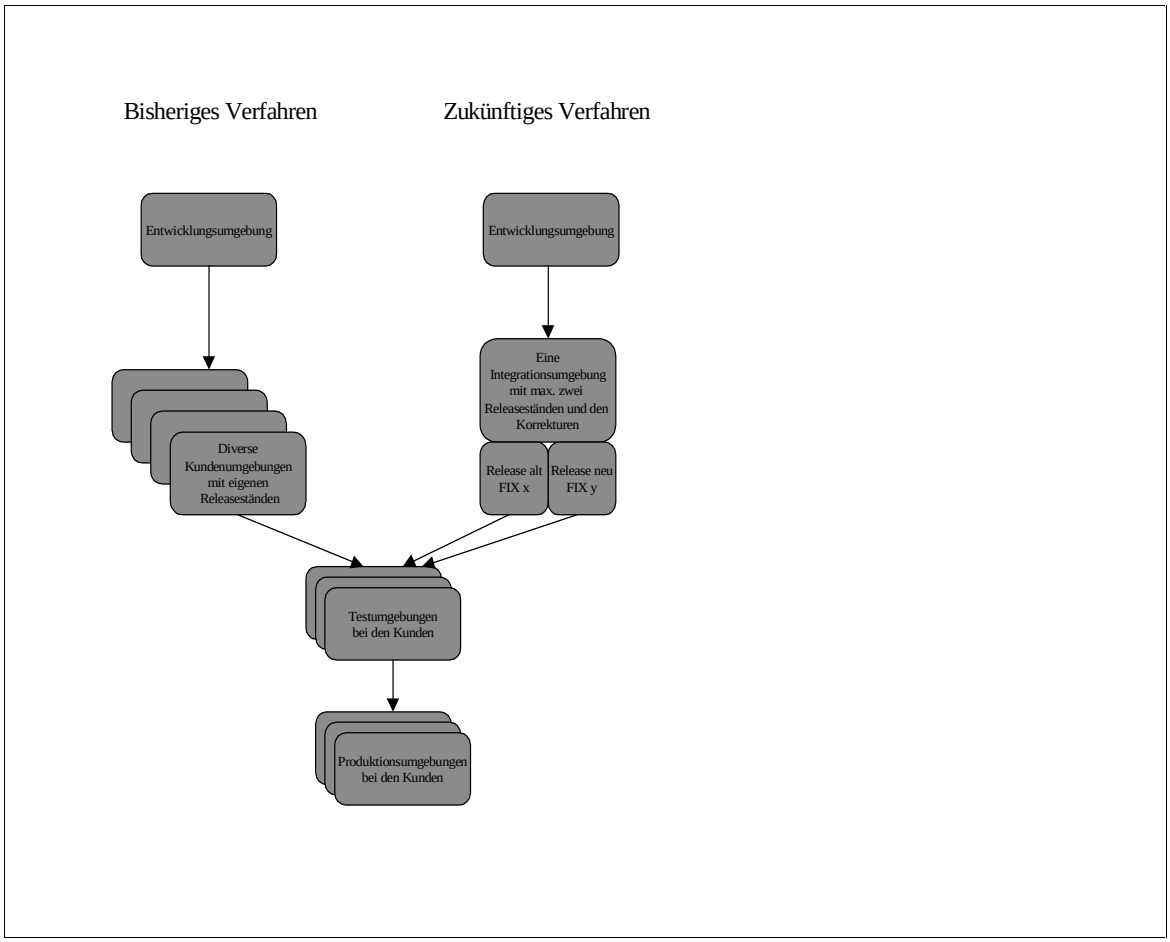


Abb. 3-2: Umgebungsmanagement

3.4 Aufteilung der Entwicklungsumgebung

Im folgenden werden die für das PRORIS-Team relevanten Bereiche und ihre Aufgaben näher beschrieben. Diese Bereiche werden bei PRORIS über lokale bzw. Netzwerklaufwerke abgebildet. Viele der verwendeten Entwicklungswerkzeuge sind nicht in der Lage modernere Verfahren wie z.B. logische Volumes oder UNC-Namen[27] zu unterstützen.

Lokale Bereiche

Auf der lokalen Festplatte werden keine wichtigen Daten abgelegt, da die lokalen Platten nicht gesichert werden. Alle Dateiserver dagegen werden jede Nacht komplett gesichert. Die Historie besteht aus 30 bzw. 31 Tagen. Beim versehentlichen Löschen von Dateien besteht somit eine gute Chance, die Daten wiederherstellen zu können.

Bei jeder Anmeldung an einer Proris-Arbeitsstation läuft ein Skript, das das Verzeichnis \proris aktualisiert. Dieses enthält Kopien von ausgewählten Dateien aus dem Entwicklungsbereich, um die Übersetzung von Programmen zu beschleunigen. Bei diesen Dateien handelt es sich im wesentlichen um Headerdateien die nur selten geändert werden. Ohne diese Maßnahmen würde sich der Übersetzungvorgang der C Quellcodedateien erheblich verlangsamen.

Firmenbereich:

Der Firmenbereich besteht aus einem Novell-Netz, das vom Proris-Team dazu genutzt wird, auf das firmenweite Transferverzeichnis zuzugreifen, um Dokumentation und Schriftverkehr abzulegen bzw. auszutauschen und mit anderen Mitarbeitern außerhalb des Proris-Teams zu kommunizieren. Auf diesen Bereich haben alle Proris-Mitarbeiter Zugriff.

Bereich für SQL-Dateien:

Dieser Bereich dient zur Ablage von SQL-Dateien. Diese Dateien werden dazu verwendet, um neue Tabellen oder Daten in der Datenbank anzulegen oder bestehende Strukturen und Daten zu ändern. Auf diesen Bereich haben alle Proris-Mitarbeiter Zugriff.

[27] UNC steht für universal naming convention

PRORIS Transfer-Bereich:

Dies ist der Transferbereich für das PRORIS-Team. Es dient zum Transfer von Daten innerhalb des PRORIS-Teams. Auf diesen Bereich haben alle Proris-Mitarbeiter Zugriff.

Bereich für Kundenumgebungen:

In diesem Bereich werden die jeweiligen Kundenumgebungen zur Verfügung gestellt. Er ist nur im Zugriff, wenn man sich mit einem Testbenutzer für den jeweiligen Kunden auf der Arbeitsstation angemeldet hat. Testbenutzer dienen dazu, die an Kunden ausgelieferte Versionen von PRORIS zu testen und warten.

In den Kundenumgebungen liegen hierbei die ausführbaren Dateien von PRORIS der jeweiligen Kundenumgebung, während der oben beschriebene Transferbereich die Konfigurationsdateien enthält, die PRORIS zum Start benötigt.

Auf die Kundenumgebung haben die Tester und Entwickler nur lesenden Zugriff. Ausschließlich die QM-Mitarbeiter können in dieser Umgebung Änderungen durchführen und z.B. neue Versionen zu Verfügung stellen. Dazu muß ein dokumentierter und nachvollziehbarer Integrationsprozeß[28] durchlaufen werden.

Bereich für die Entwickler:

In diesem Bereich befindet sich für jeden Entwickler die persönliche Entwicklungsumgebung. In dieser Umgebung werden die Entwicklungs-Wartungsaufgaben vom jeweiligen Entwickler durchgeführt. Jeder Entwickler hat dort seine eigene Umgebung.

Bei Auschecken von Dateien mit dem PVCS Version-Manager werden die auf dem zentralen Entwicklungslaufwerk befindlichen Dateien auf gleichlautende Verzeichnisse innerhalb der jeweiligen Entwicklerumgebung kopiert.

Bereich für Entwicklungsprodukte und Werkzeuge:

Dieser Bereich enthält die Installation und die Konfigurationsdateien der SKM Werkzeuge, insbesondere den PVCS Version Manager, PVCS Configuration Builder und den PVCS Tracker. Darüber hinaus sind in diesem Bereich diverse Eigenentwicklungen sowie Fremdprodukte zur Entwicklungsunterstützung zu finden.

[28] siehe auch Abschnitt 5.2.6

Auf diesen Bereich haben alle Proris-Mitarbeiter Zugriff.

Zentraler Entwicklungsbereich:

Der zentrale Entwicklungsbereich enthält alle Archive des Version Managers, die Tip Revisions aller PRORIS-Quelltexte, alle Zwischendateien und ausführbaren Dateien. Alle Dateien in diesem Bereich sind schreibgeschützt. Änderungen können ausschließlich mit den PVCS Werkzeugen vorgenommen werden. Damit ist die Integrität der Dateien in diesem Bereich gewährleistet. Auf diesen Bereich haben alle Proris-Mitarbeiter Zugriff.

3.5 Datensicherung

Alle PRORIS Serversysteme werden täglich vollständig über ein Bandlaufwerk gesichert. Die Historie der Sicherungsbänder beträgt einen Monat. Danach werden sie wieder überschrieben. Die Bänder werden in einer feuersicheren Box an einem sicheren Ort in der Firma aufbewahrt. Einmal in der Woche wird ein Sicherungssatz, bestehend aus der PRORIS-Sicherung und den anderen Sicherungsbändern der Firma, in einem Bankschließfach verwahrt. Zusätzlich wird eine Sicherung eines jeden beim Kunden ausgelieferten Releasestands von PRORIS in dem Bankschließfach verwahrt, solange bis die entsprechende Version bei keinem Kunden mehr im Einsatz ist.

Die Sicherungsbänder werden in regelmäßigen Abständen durch frische Bänder ersetzt. Darüber hinaus wird die ordnungsgemäße Funktion der Sicherungsgeräte durch Prüfroutinen und Stichproben überwacht.

Zu dem Sicherungssatz in der PRORIS Umgebung gehören alle Datenbanken sowie alle Netzlaufwerke[29]. Zusätzlich werden alle Systempartitionen der OS/2- und Windows NT Server gesichert. Damit ist, im Falle eines Ausfalls, eine schnelle Wiederherstellung eines oder mehrerer Systeme gewährleistet.

Die Entwicklungsrechner werden nicht gesichert, da alle für die Entwicklung entscheidenden Dateien auf den Dateiservern liegen und somit der zentralen Datensicherung unterliegen. Damit beim Ausfall oder einer Neuinstallation schnell reagiert werden kann, liegt von der unter 3.2 beschriebenen Standardinstallation ein Sicherungsabbild vor, mit dessen Hilfe ein Entwicklungsrechner schnell wiederhergestellt werden kann.

[29] siehe Abschnitt 3.4

3.6 Bewertung der Entwicklungsumgebung

Die folgende Bewertung und auch alle weiteren in den nächsten Kapiteln erfolgen aus Sicht des Autors. Da der Autor überwiegend im Bereich Qualitätssicherung tätig ist, erfolgt die Bewertung aus Sicht des Qualitätsmanagements.

Ein Aspekt der bei der Betrachtung der PRORIS-Entwicklungsumgebung sofort auffällt, ist deren Komplexität. Dies begründet sich im wesentlichen aus zwei Punkten. Zum einen die Notwendigkeit zwei Betriebssysteme zu unterstützen und zum anderen die unterschiedlichen Datenbankversionen, die bei den Kunden zum Einsatz kommen.

Erschwerend kommt hinzu, daß das Betriebssystem OS/2 nicht mehr von den Softwareherstellern unterstützt wird und damit keine modernen Werkzeuge unter OS/2 zur Verfügung stehen. Dieses bewirkt für das Änderungsmanagementsystem PVCS Tracker, den Einsatz einer veralteten Version, die wesentliche Verbesserungen im Änderungsmanagement und bei der Integration mit dem Versionsmanagement nicht beinhaltet.

Der Ansatz OS/2 als Betriebssystem zu unterstützen solange es noch bei Kunden im Einsatz ist, kann als vermeintlich kundenfreundlich bezeichnet werden. Auf der anderen Seite hemmt es Neuentwicklungen und verschlechtert somit ggf. die Marktposition des Produkts. Durch die Unterstützung von OS/2 als Plattform werden Ressourcen gebunden, die für die Weiterentwicklung des Produktes wertvoll sind.

Der Einsatz von drei verschiedenen Datenbankversionen bei den Kunden begründet sich durch die häufigen Releasewechsel bei der IBM Datenbank. Viele Kunden sehen sich nicht in der Lage diese Releasewechsel zeitnah in ihrer Umgebung durchzuführen. Andere sind Aufgrund von Vorschriften dazu gezwungen. Für ein Softwarehaus besteht nun die Aufgabe darin in diesem Spannungsfeld einen geeigneten Mittelweg zu finden, der die Interessen möglichst vieler Beteiligter berücksichtigt, ohne die eigenen Interessen dabei zu vernachlässigen.

Ein weiteres Problem stellt die Tatsache dar, daß das Änderungsmanagementsystem PVCS Tracker, seine Daten nicht in der IBM Datenbank speichern kann, die im PRORIS-Umfeld verwendet wird. Dadurch wird die Komplexität des Gesamtsystems um ein weiteres Datenbankmanagementsystem, in diesem Fall Sybase, erhöht.

Gleiches gilt für die Compiler, die unter OS/2 und Windows NT nicht vom gleichen Hersteller kommen. Die Unterschiede in diesem Bereich bleiben dem Entwickler durch eine gute Integration mit dem Build-Management und den Einsatz einer einheitliche IDE verborgen. Dies geht zu Lasten des Komforts, da somit nur die relativ alte Workbench von Microsoft eingesetzt werden kann.

Ein weiterer Punkt ist die Vorhaltung lokaler Kopien ausgewählter Konfigurationselemente (Header-Dateien). Durch diese Redundanz besteht potentiell die Gefahr von Inkonsistenzen zwischen zentraler Datei und der lokalen Kopie. Der hohe Geschwindigkeitsgewinn in der täglichen Arbeit und organisatorische Maßnahmen zur Eindämmung von Inkonsistenzen, rechtfertigen jedoch nach Ansicht des Autors die lokalen Kopien.

Durchweg positiv zu bewerten ist die zentrale Installation und Konfiguration der Entwicklungswerkzeuge. Der Entwickler muß sich somit nicht mehr mit seiner Installation oder mit Einstellungen in der Entwicklungsumgebung beschäftigen, sondern kann sich auf die Entwicklungsaufgabe konzentrieren.

Ein weiter positiver zu hervorhebender Punkt ist das umfangreiche Datensicherungskonzept, das die Wichtigkeit der Daten in der Entwicklungsumgebung für ein Softwarehaus und seine Kunden dokumentiert.

4 Aufbau des PRORIS Änderungsmanagement

Zu Beginn dieses Kapitels wird der Entwicklungsprozeß von PRORIS näher betrachtet, da das Änderungsmanagement in diesen eingebettet ist. Dazu werden zunächst grundsätzliche Formen von Entwicklungsprozessen betrachtet und im Anschluß auf die spezifischen Gegebenheiten im PRORIS-Projekt eingegangen.

4.1 Entwicklungsprozesse

Wenn man sich mit der Frage nach einem Vorgehen für die Entwicklung von Software beschäftigt, so stellt man fest, daß es kaum einen Unterschied zu den Aspekten, die bei den sogenannten Geschäftsprozessen relevant sind, gibt. Von Interesse sind somit auch die entscheidenden Fragen der Geschäftsprozeßmodellierung [Oestereich 99]:

- In welche Teile kann der Gesamtprozeß zergliedert werden ?
- In welcher Reihenfolge können/müssen diese Teilschritte ausgeführt werden ?
- Welche Ressourcen (Personen und Informationen) sind erforderlich ?
- Welche Informationen müssen als Eingangsgrößen vorliegen ?
- Welche Ergebnisse werden von welchen Teilschritten erzeugt ?

Eine Klassifizierung unterschiedlicher Vorgehensmodelle ist nach vielerlei Kriterien möglich. Grundsätzlich lassen sie sich nach der zugrunde liegenden Vorgehensweise einordnen. Die althergebrachten Prozeßmodelle, allen voran das Wasserfallmodell, sind aufgrund der vorweggenommenen Planung sequentiell ausgerichtet. Erkenntnisse, die sich im Laufe des Projektes ergeben, erzwingen jedoch einen höheren Grad an Zusammenarbeit über alle Phasen eines Projektes hinweg und führen somit zu einer inkrementellen und iterativen Vorgehensweise.

Im Wasserfallmodell geht man davon aus, daß durch eine ausgedehnte und detaillierte Analysephase den potentiellen Veränderungen, die sich im Laufe der meisten Projekte ergeben entgegengewirkt werden könne. Bei einer inkrementellen Vorgehensweise nimmt man solche Veränderungen der Anforderungen in Kauf und entwickelt daher das System in mehreren Durchläufen, um in späteren Inkrementen den Veränderungen Rechnung zu tragen.

Es werden im folgenden einige typische Vorgehensweisen vorgestellt:

4.1.1 Kein explizit formulierter Prozeß

Keinen expliziten Entwicklungsprozeß zu haben, ist in vielen Projekten heute noch der Normalfall (Stand 1998: ca. 68 % aller Entwicklungsprojekte). Nach einer Anforderungsanalyse, wird solange programmiert, getestet und korrigiert, bis die Software einen akzeptablen Stand erreicht hat. Es liegt eine ad hoc Vorgehensweise vor, der keine genaue Planung und keine gezielte Abfolge von Schritten zugrunde liegt.

Vorteile:

- Der Prozeß hat kaum Overhead (Design, Dokumentation, Einhalten von Standards)
- Jeder kann den Prozeß anwenden, so daß keinerlei Vorkenntnisse erforderlich sind.

Nachteile:

- Es gibt keine Möglichkeit abzuschätzen, welchen Stand die Softwareentwicklung erreicht hat
- Risiken lassen sich nicht vorhersagen
- Die Instrumente der Projektsteuerung können nicht greifen
- Der Overhead kommt ggf. durch aufwendige nachträgliche Abstimmungen

Software ohne Prozeß zu entwickeln, funktioniert heute im Prinzip nur noch bei kleinen, wenig komplexen Projekten.

4.1.2 Wasserfallprozeß

Der Wasserfallprozeß ist einer der bekanntesten und am häufigsten eingesetzten Prozesse. Wie in Abbildung 4-1 dargestellt, verläuft die Entwicklung in definierten Phasen, die sich nicht überlappen. Zunächst werden die Anforderungen betrachtet und in ein Analysemodell umgesetzt. In der Entwurfsphase wird dann entschieden, wie das Analysemodell realisiert werden soll. Dieses wird dann programmiert, integriert und getestet. Der Prozeß hat somit klar definierte Phasen. Zum Ende jeder Phase gibt es ein Review, in dem entschieden wird, ob die Ziele der Phase erreicht worden sind. Erst wenn eine Phase erfolgreich abgeschlossen wurde, kann der Übergang zur nächsten Phase erfolgen. Die Rückkehr in vorherige Phasen ist aufwendig (weiße Pfeile) und nicht oder unzureichend definiert. Es wird vielmehr davon ausgegangen, daß ein solcher Rückschritt nicht notwendig ist.

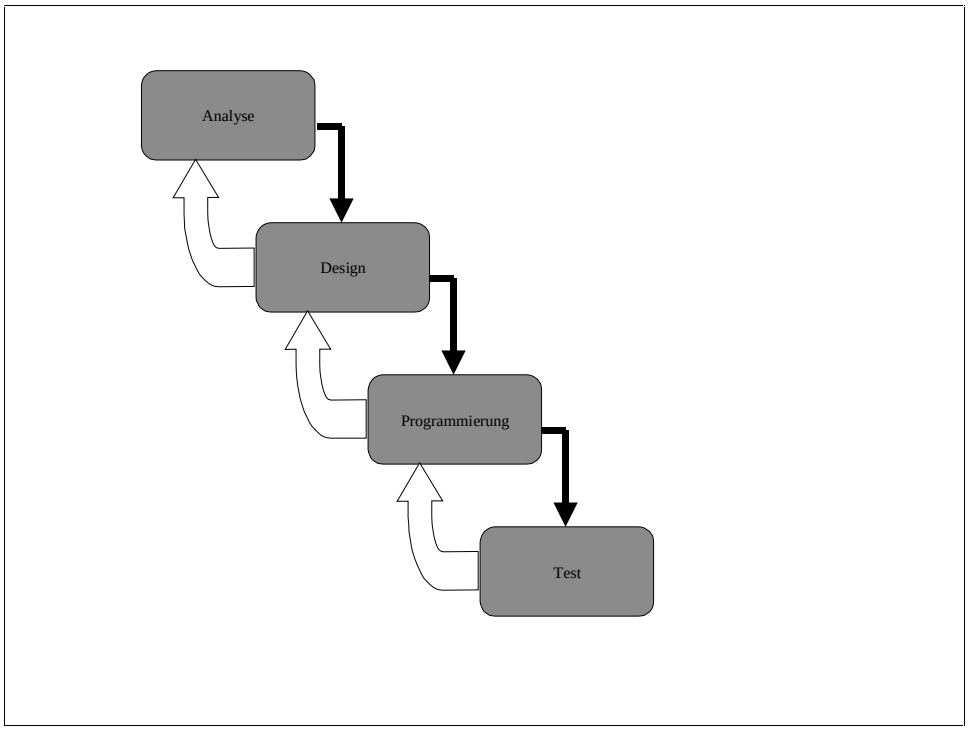


Abb. 4-1: Phasen eines typischen Wasserfallprozesses

Vorteile:

- Aufgaben der Phasen sind klar beschrieben. Grobe Fehler können bereits in frühen Phasen entdeckt werden.
- Der Prozeß ist für die Teammitglieder leicht zu erlernen, da es klare Anweisungen gibt, was in jeder Phase zu tun ist.
- Der Overhead hält sich in Grenzen. Trotzdem können die Aufgaben einer jeden Phase genau geplant werden. Projekte lassen sich somit bereits steuern.

Nachteile:

- Der Wasserfallprozeß geht davon aus, daß alle Anforderungen bekannt und im wesentlichen stabil sind.
- Die Komplexität des Aufgabenumfelds beherrschbar ist und somit die Entwicklung in einem Inkrement durchführbar ist.
- Risiken werden erst spät im Projektverlauf erkannt.

Gerade bei umfangreichen Neuentwicklungen treffen die beiden ersten Punkte nicht zu. Die Anforderungen stabilisieren sich erst im Projektverlauf. Manche Probleme treten erst bei der Programmierung zu Tage. Häufig verändert sich auch das Projektumfeld, nicht

zuletzt aufgrund neuer Technologien. Dies führt zu neuen oder geänderten Anforderungen, die nachträglich in das Projekt eingebaut werden müssen. In der Praxis ergeben sich viele Probleme erst während der Implementierung oder der Systemintegration. Es wird festgestellt, daß Schnittstellen nicht zusammenpassen oder die Performance nicht ausreichend ist. Damit eignet sich der Wasserfallprozeß nicht für ein risikobewußtes Vorgehen, weil viele Risiken erst in den späten Phasen erkannt werden können.

4.1.3 Der iterativ-inkrementelle Entwicklungsprozeß

Die iterative und inkrementelle Vorgehensweise bei der Entwicklung von Software ist die Antwort auf viele Probleme des Wasserfallprozesses. Statt zu versuchen das System in einem Rutsch zu erstellen, wird die Anwendung beim iterativen Vorgehen schrittweise entwickelt. Bei dieser Vorgehensweise wird von vornherein angenommen, daß:

- die Anforderungen unvollständig sind
- sich Anforderungen während des Projektes teilweise ändern können
- sich Anforderungen aufgrund von gewonnenen Erkenntnissen im Laufe der Zeit vervollständigen.

Der Prozeß zielt darauf ab, diese Änderungen und deren Auswirkungen kontrolliert und nicht chaotisch in das Projekt einzubringen. Er erlaubt dem Management damit eine sehr viel feinere Steuerung des Ablaufs, als dies mit dem Wasserfallprozeß möglich ist.

Begriffsbestimmung:

Iteratives Vorgehen:
ein Vorgehen, bei dem der Entwicklungsprozeß in mehrere gleichartige Zeitabschnitte (Iterationen) geteilt wird. Die Schritte (Aktivitäten), die in jeder Iteration durchgeführt werden, sind in jeder Iteration dieselben. Jede Iteration erzeugt ein Teilergebnis.

inkrementelles Vorgehen:
eine Vorgehensweise, bei der ein Produkt schrittweise in wachsenden Zwischenprodukten entsteht.

Die Vorteile dieses Prozesses liegen auf der Hand:

- flexibler Umgang mit Änderungen der Anforderungen und Rahmenbedingungen, d.h. Anforderungen die zu einem späteren Zeitpunkt auftreten, können geordnet berücksichtigt werden. Neue Anforderungen müssen durch ein CCB genehmigt werden

und werden dann, nach Prioritäten geordnet, in den folgenden Iterationen berücksichtigt.

- umfassendes und objektiveres Projektcontrolling:
  der Projektfortschritt wird sichtbar, da das Projektteam gezwungen ist, regelmäßig neue Versionen des Systems fertigzustellen, d.h. der Projektfortschritt kann anhand lauffähiger Programme kontrolliert werden. Probleme mit dem Zeitplan werden somit rechtzeitig erkannt und entsprechende Gegenmaßnahmen können zeitnah eingeleitet werden.
- effektiveres Risikomanagement:
  da jede Iteration zu einem lauffähigen System führt, können unvollständige Anforderungen und andere Probleme sehr viel früher erkannt werden. Ziel ist es kritische Systemteile möglichst früh zu erkennen und zu realisieren.
- fortlaufende Projektplanung:
  der Prozeß ermöglicht die Projektplanung nach jeder Iteration zu verfeinern und verbesserte Aufwandsschätzungen bereitzustellen.
- effektiveres Testen:
  das Produkt wird nicht erst kurz vor der Einführung umfassend getestet, sondern nach jeder Iteration. Das Projektteam erhält dadurch frühzeitig Rückmeldung über vorhandene Probleme, da ein lauffähiges Zwischenprodukt durch den Endbenutzer in Augenschein genommen werden kann.
- flexible Auslieferung:
  mit dem Kunden können , wenn er dieses wünscht, vorzeitig lauffähige Versionen des Systems zu Verfügung gestellt werden, die schon eine definierte Teilfunktionalität aufweisen.
- desweiteren ergeben sich Vorteile die einen eher qualitativen Charakter besitzen und demzufolge schwer meßbar sind , wie z.B.:
  - eine höhere Akzeptanz bei den Benutzern und Auftraggebern
  - höhere Motivation und Ergebnisorientierung bei den Entwicklern.

Neben den genannten Vorteilen sollen aber nicht der erhöhte Aufwand bei der Planung und Steuerung des Projekts und die höheren Anforderungen an das Projektmanagement verschwiegen werden.

Eines der bekanntesten iterativen Verfahren ist das Spiralmodell von Barry Boehm [Boehm 88]. Bei jeder Umdrehung der Spirale, die einer Iteration entspricht, werden kritische Systemteile angegangen und realisiert. Erst wenn dieser Teil funktioniert, wird die nächste

Iteration geplant. Bei diesem Modell wird strikt nach Risikogesichtspunkten vorgegangen. D.h., daß die größten Risiken in einer Reihe von Iterationen analysiert und bewertet werden. Im Anschluß kann das Projekt dann z.B. mit einem Wasserfallprozeß fortgeführt werden.

4.1.4 Der Entwicklungsprozeß bei PRORIS

Da PRORIS-System ist Anfang der neunziger entworfen worden. Die Technologien die zum Einsatz kommen sollten (Client/Server System mit relationaler Datenbank und grafischer Benutzeroberfläche) waren für damalige Verhältnisse Neuland. Die fachlichen Fragen, die von dem System gelöst werden sollten, waren überwiegend aus vorherigen Entwicklungen, wenn auch nicht in dem geforderten Umfang, bekannt. Insofern gab es für den fachlichen und den technischen Teil des PRORIS-Systems unterschiedliche Ansätze.

Da der fachliche Teil als bekannt und mit wenig Risiken behaftet zu sein schien, wurde hierfür der Wasserfallprozeß zur Anwendung gebracht. Im Rahmen eines Pflichtenheftes und Sollkonzeptes, wurde zu Beginn versucht, das komplette System mit allen Abhängigkeiten und Schnittstellen zu beschreiben.

Da im technischen Teil die größten Risiken zu finden waren, wurde hier eher iterativ vorgegangen, indem die technischen Risiken wie z.B. die Datenbankanbindung, die Steuerung der grafischen Benutzeroberfläche oder auch die Performance anhand einer Beispielanwendung getestet wurde.

Inzwischen kann man das Projekt PRORIS als Kombination von Wartungsaktivitäten und Neuentwicklung bezeichnen. Da PRORIS ein Produkt darstellt, das an neue Kunden verkauft wird, unterliegt es andauernd neuen oder veränderten Anforderungen, die in den Entwicklungsprozeß eingebracht werden müssen. Auf der anderen Seite muß die bestehende Funktionalität gewartet werden. Der Entwicklungsprozeß ist somit iterativ, wobei sich die Iterationen in neuen Releaseständen widerspiegeln.

4.2 Änderungsmanagement bei PRORIS

4.2.1 Der PRORIS-Änderungsprozeß

In PRORIS werden alle Änderungsanforderungen in Form von PRORIS-Meldungen[30] erfaßt. Dies gilt sowohl für externe (vom Kunden) als auch interne Anforderungen bzw. Fehlermeldungen. Zur Dokumentation und Verfolgung dieser Meldungen wird PVCS

[30] siehe Anhang B

Tracker[31] eingesetzt. Dieses Werkzeug ermöglicht die Erfassung, Verfolgung und Auswertung der PRORIS-Meldungen.

Die Bearbeitung der PRORIS-Meldungen wird Abbildung 4-2 dargestellt.

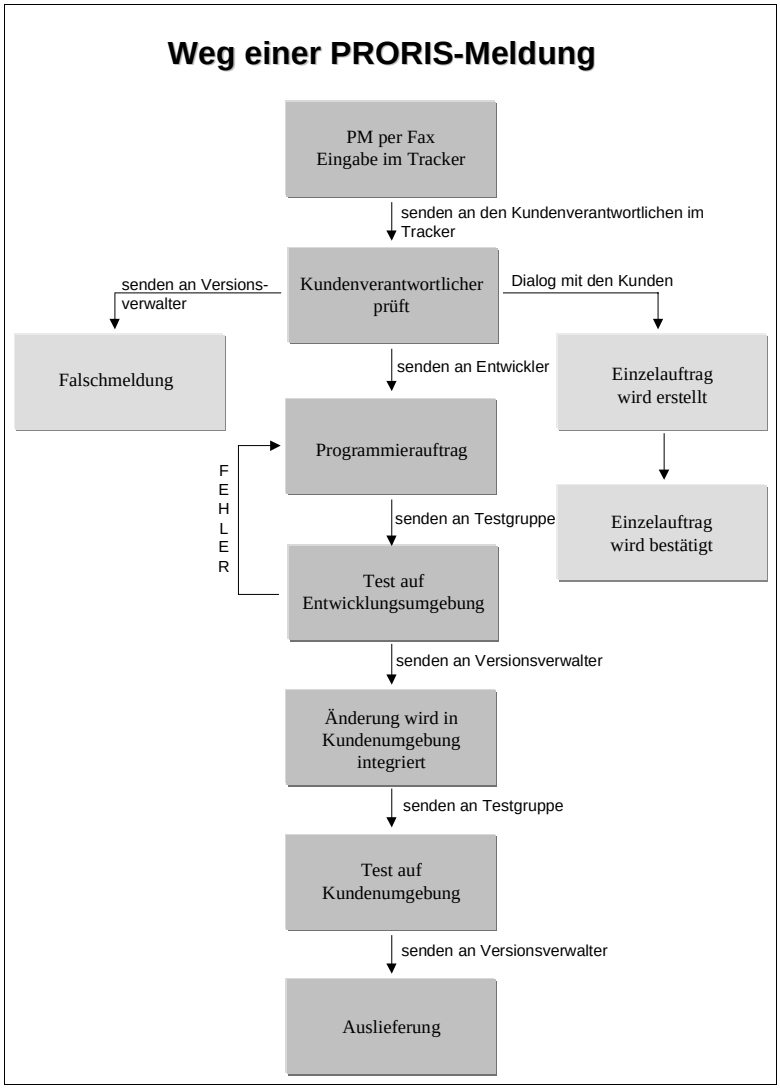


Abb. 4-2: Weg einer PRORIS-Meldung

[31] siehe Abschnitt 4.2.2

Es folgt eine Beschreibung der einzelnen Aktivitäten bei der Meldungsbearbeitung:

Schritt 1:

Ankommende PRORIS-Meldung (meist per Fax) wird von der Projektassistenz bei der CTH in PVCS-Tracker erfaßt. Die Folgenden Felder werden bei diesem Arbeitsschritt belegt.

Die **'Beschreibung'** (Beschreibungstext) wird vollständig in der Tracker-Datenbank erfaßt.

Der **'Kunde'** wird mit einer Combobox ausgewählt. Die Kunden sind Schlüsseln zugeordnet. Jeder neue Kunde muß in der Trackerdatenbank als Schlüssel angelegt werden.

Das **'Meldungsdatum'** wird von der PRORIS-Meldung übernommen.

Die **'Meldungsnummer'** (Lfd.-Nr.) wird ebenfalls aus der Meldung übernommen.

Die **'Nachtragsnummer'** muß belegt werden (Wert >= 1). Es dient dazu, einen Nachtrag zu einer schon vorher eingegangenen Meldung zu identifizieren.

Der **'Meldungstyp'** (Typ) wird über eine Combobox ausgewählt. Die möglichen Typen sind Fehler, Verbesserung, Erweiterung, Prüfung, Falschmeldung und Angebot. Eine Falschmeldung ist eine Fehlermeldung, bei der sich herausstellt, das es sich um kein Fehler des Systems handelt.

Die **'Priorität'** gibt an, bis zu welchem Zeitpunkt sich die CTH zurückgemeldet haben soll. Insofern muß vom jeweils für die PRORIS-Meldung Zuständigen (zum Zeitpunkt der Erfassung die Projektassistenz) geprüft werden, ob der Termin eingehalten werden kann. Das bedeutet, daß, wenn kein Kundenbetreuer bis zu dem geforderten Termin in der Lage ist, Rückmeldung zu geben, der Erfasser Rückmeldung geben muß. Mit anderen Worten, die Meldung bleibt solange in der Verantwortung des Bearbeitenden, bis der nächste in der Bearbeitungskette die Meldung übernommen hat (=> siehe Bearbeitungsstatus). Die Rückmeldung kann auch daraus bestehen, daß bis zu einem bestimmten Termin keine qualifizierte Rückmeldung z.B. wegen Krankheit erfolgen kann. Es folgt eine Aufstellung der möglichen Prioritäten:

showstopper	Sofortige Rückmeldung (Der Betrieb kann nicht fortgesetzt werden) => **nur bei Meldungstyp Fehler**
hoch	Rückmeldung am gleichen Arbeitstag wenn die Meldung bis 13.00 Uhr eingegangen ist. Bei Eingang nach 13.00 Uhr wird am nächsten Arbeitstag reagiert.
mittel	Rückmeldung innerhalb von 5 Arbeitstagen

niedrig	Rückmeldung innerhalb von 10 Arbeitstagen
keine	irrelevant

Der Rückmeldungstermin ist im Feld **'Rückmeldung am'** einzutragen. Dieser errechnet sich aus dem Meldungsdatum und der Priorität.

Das Feld **'Wo gefunden'** beschreibt die Umgebung (Test, Produktion, etc.), auf die sich die Meldung bezieht.

Das Feld **'Ansprechpartner'** wird nur wenn die Meldung vom Kunden kommt gefüllt. Es beschreibt diejenige Person bei dem Kunden, die zu der Meldung nähere Auskünfte geben kann.

Das Feld **'Anwendung'** benennt den Teil des PRORIS-Systems, in dem der Fehler bzw. der Verbesserungsbedarf gefunden wurde. Dieses Feld orientiert sich an der PRORIS-Produktstruktur[32].

Die **'gewünschte Fertigstellung'** wird aus der PRORIS-Meldung übernommen.

Das Feld wird nur gefüllt, wenn die Meldung vom Kunden kommt.

Der **'Bearbeitungsstatus'** wird auf den Wert **'Neu'** gesetzt. Das bedeutet gleichzeitig, daß der Erfasser der Meldung, für diese solange zuständig ist, bis der Kundenbetreuer die Bearbeitung übernimmt (=> siehe Feld: Priorität).

Das Feld **'Meldung gesendet an'** wird jeweils vom Meldungsbearbeiter auf den Meldungsempfänger umgesetzt, wenn der Meldungsbearbeiter die Meldung weiterreicht. Bei diesem Arbeitsschritt (Schritt 1), wird der jeweilige Kundenbetreuer als Meldungsempfänger eingetragen. Dieses Feld soll den Meldungsfluß in der Bearbeitungskette dokumentieren. Somit können z.B. liegengelassene Meldungen identifiziert werden.

Im Feld **'Kundenbetreuer'** wird der jeweils für einen Kunden zuständige Kundenbetreuer eingetragen.

Das Feld **'nächste Aktion durch'** sagt aus, ob zu einem bestimmten Zeitpunkt die CTH oder der Kunde für die Weiterbearbeitung der Meldung verantwortlich ist. Zum Zeitpunkt der Erfassung ist in das Feld der Wert **'CTH'** einzutragen.

[32] siehe auch Abschnitt 5.1.1

Nachdem die PRORIS-Meldung erfaßt wurde, wird die Meldung automatisch in den Posteingangskorb des Erfassers gestellt, damit er die Meldung an den Kundenbetreuer senden kann.

Dies entfällt, wenn der Erfasser und Kundenbetreuer die gleiche Person sind.

Die Meldung kann nun aus dem Posteingangskorb entfernt werden. Sie muß aber vom Erfasser noch verfolgt werden, bis der Kundenbetreuer den Bearbeitungsstatus auf 'Kundenbetreuer prüft' umgesetzt hat. Denn solange ist der Erfasser noch für die Meldung verantwortlich und muß prüfen, ob der Rückmeldetermin (Rückmeldung am) eingehalten werden kann. Zur Prüfung, für welche Meldungen der Erfasser noch zuständig ist, kann ein entsprechender Report verwendet werden.

Schritt 2:

Der Kundenbetreuer nimmt die Meldung entgegen (übernimmt die Verantwortung), wenn er den **'Bearbeitungsstatus'** auf **'Kundenbetreuer prüft'** setzt. Damit wird der Kundenbetreuer auch für die Einhaltung der Rückmeldefrist verantwortlich.

Der Kundenbetreuer klärt nun, welche Fragen in Zusammenhang mit der Meldung noch gelöst werden müssen. Dabei kann es vorkommen, daß der Kunde aktiv werden muß und damit die Verantwortung für die Weiterbearbeitung der Meldung an den Kunden übergeht. Der Kundenbetreuer trägt in diesem Fall in dem Feld **'nächste Aktion durch'** den Wert Kunde ein. Im Feld **'Rückmeldung am'** wird die mit dem Kunden abgestimmte nächste Kontaktaufnahme vereinbart. Gleiches gilt für den Fall, daß der Kundenbetreuer bis zu einem bestimmten Termin Fragen zur Meldung klären soll (**'nächste Aktion durch'** bleibt auf **'CTH'**).

Im Rahmen der Prüfung kann in Absprache mit dem Kunden der Meldungstyp umgesetzt werden. So kann z.B. aus dem **'Meldungstyp' 'Fehler'** bei einer Falschmeldung der **'Meldungstyp' 'Falschmeldung'** werden. Die Dokumentation der Abstimmgespräche erfolgt in Kurzform unter der Meldungsnotiz '**Bearbeitungsstand**'.

Wenn die Meldung über längere Zeit nicht weiterbearbeitet werden soll (in Abstimmung mit dem Kunden), wird in dem Feld **'Lösung'** der Wert **'verschoben'** eingetragen. Die Meldung wird dann zum Benutzer "Benutzertreffen", kopiert. Das Kopieren unterscheidet sich vom Verschieben dadurch, daß die Meldung nicht vom eigenen InTray gelöscht wird. Auf dem Benutzertreffen kann dann entschieden werden, was mit dieser Meldung weiter geschehen soll.

Bei einer Verbesserung oder einer Erweiterung setzt der Kundenbetreuer, nachdem er dem Kunden per PRORIS-Einzelauftrag ein Angebot geschickt hat, den **'Bearbeitungsstatus'** auf den Wert **'Angebot'**. Somit kann der Kundenbetreuer prüfen auf welche Angebote vom Kunden noch nicht reagiert worden ist.

Mit dem Kunden ist ein Rückmeldungstermin (Feld **'Rückmeldung am'**) zu vereinbaren, an dem die CTH oder der Kunde wieder aktiv wird. Der Realaufwand für die Prüfung, also der Aufwand des Kundenbetreuers bis zu dem Zeitpunkt an dem entschieden wurde, daß die Weiterbearbeitung der Meldung vorschoben wird, wird in dem Feld **'Real Konzept'** eingetragen. Bei Wiederaufnahme der Meldung gelten für die Weiterbearbeitung der Meldung die Regeln der verschiedenen Meldungstypen.

Die Prüfung der Meldung muß abhängig vom Meldungstyp folgende Ergebnisse liefern:

Meldungstyp Fehler:

Es wurde geprüft, in welcher Version der Fehler gefunden wurde (Feld **'gefunden in Build'**).

Wie der Fehler reproduziert werden kann, wurde unter der Meldungsnotiz **'Wie wird der Fehler reproduziert'** dokumentiert (z.B. bestimmte Datenkonstellationen).

Eine Beschreibung, wie die Behebung des Fehlers später zu testen ist, wurde unter der Meldungsnotiz **'Testanleitung'** erfaßt.

Wenn eine Umgehung des Fehlers möglich ist, wurde diese unter der Meldungsnotiz **'Umgehung des Fehlers'** beschrieben.

Der Lösungsansatz für die Behebung des Fehlers wurde unter der Meldungsnotiz **'Lösungsansatz'** dokumentiert. Dies ist gleichzeitig die Vorgabe für den Entwickler.

Ein vorhandener Bezug zu einer anderen Meldung wurde unter der Meldungsnotiz **'Bezug zu anderen Meldungen'** erfaßt.

Der Entwickler, der die Meldung bearbeiten soll, wurde in das Feld **'Entwicklername'** eingetragen.

In dem Feld **'Lösung'** wurde der Wert **'Programmänderung'** eingetragen.

Der mit dem Kunden abgestimmte Auslieferungstermin für die Fehlerkorrektur wurde in dem Feld **'geplante Auslieferung'** erfaßt.

Der **‘Projekt’**-, **‘Tätigkeit’**- und **‘Faktura’**-schlüssel wurde in die entsprechenden Felder eingetragen. Diese Felder dienen dem Projektcontrolling.

Die Planaufwände (**‘Plan Konzept’**, **‘Plan Entwicklung’**, **‘Plan Test’, ‘Plan Dokumentation’**, **‘Plan gesamt’**) sind in die dafür vorgesehen Felder eingetragen worden.

Der Realaufwand für die Prüfung, also der Aufwand des Kundenbetreuers bis zum Ende des zweiten Schrittes wurde in dem Feld **‘Real Konzept’** eingetragen.

Dokumente, die im Rahmen der Prüfung erstellt wurden, sind mit der Meldung verknüpft worden.

Der Kundenbetreuer reicht die Meldung an den Entwickler weiter, indem er in dem Feld **‘Meldung gesendet an’** den Entwickler einträgt und ihm die Meldung via Posteingangskorb zusendet. Danach kann die Meldung aus dem Posteingangskorb des Kundenbetreuers entfernt werden. Zur Kontrolle, welche Meldungen des Fachbetreuers der Entwickler noch nicht bearbeitet hat, kann ein entsprechender Report verwendet werden.

=> weiter mit Schritt 3

Meldungstyp Verbesserung/Erweiterung/Angebot:

Der Lösungsansatz wurde unter der Meldungsnotiz **‘Lösungsansatz’** dokumentiert. Dies ist gleichzeitig die Vorgabe für den Entwickler.

Ein vorhandener Bezug zu einer anderen Meldung wurde unter der Meldungsnotiz **‘Bezug zu anderen Meldungen’** erfaßt.

Der Entwickler wurde in das Feld **‘Entwicklername’** eingetragen.

In dem Feld **‘Lösung’** wurde der Wert **‘Programmänderung’** eingetragen.

Der mit dem Kunden abgestimmte Auslieferungstermin für die Verbesserung/Erweiterung wurde in dem Feld **‘geplante Auslieferung’** erfaßt.

Der **‘Projekt’**-, **‘Tätigkeit’**- und **‘Faktura’**-schlüssel wurde in die entsprechenden Felder eingetragen.

Die Planaufwände (**‘Plan Konzept’**, **‘Plan Entwicklung’**, **‘Plan Test’, ‘Plan Dokumentation’**, **‘Plan gesamt’**) sind in die dafür vorgesehen Felder eingetragen worden.

Der Realaufwand für die Prüfung, also der Aufwand des Kundenbetreuers bis zum Ende des zweiten Schrittes, wurde in das Feld **‘Real Konzept’** eingetragen.

Ein PRORIS-Einzelauftrag wurde an den Kunden geschickt und liegt vom Kunden unterschrieben vor.

Dokumente, die im Rahmen der Prüfung erstellt wurden, sind mit der Meldung verknüpft worden.

Der Kundenbetreuer reicht die Meldung an den Entwickler weiter, indem er in dem Feld **‘Meldung gesendet an’** den Entwickler einträgt und ihm die Meldung via Posteingangskorb zusendet. Danach kann die Meldung aus dem Posteingangskorb des Kundenbetreuers entfernt werden. Zur Kontrolle, welche Meldungen des Fachbetreuers der Entwickler noch nicht bearbeitet hat, kann ein entsprechender Report verwendet werden.

=> weiter mit Schritt 3

Meldungstyp Prüfung:

In dem Feld **‘Lösung’** wurde der Wert **‘telef. Klärung’** oder **‘nicht beheben’** eingetragen.

Der **‘Projekt’**-, **‘Tätigkeit’**- und **‘Faktura’**-schlüssel wurde in die entsprechenden Felder eingetragen.

Der Realaufwand für die Prüfung wurde in dem Feld **‘Real Gesamt’** eingetragen.

Dokumente, die im Rahmen der Prüfung erstellt wurden, sind mit der Meldung verknüpft worden.

Da die Meldung erledigt ist, muß der Kunde nur noch eine Abnahmeerklärung schicken, mit der die Meldung dann geschlossen werden kann. Der Kundenbetreuer sendet die Meldung via Posteingangskorb an den Projektkoordinator, der für die Prüfung und Verfolgung der Abnahmeerklärungen zuständig ist. Danach kann die Meldung aus dem Posteingangskorb des Kundenbetreuers entfernt werden.

=> Ende

Meldungstyp Falschmeldung:

In dem Feld **‘Lösung’** wurde der Wert **‘Falschmeldung’** oder **‘doppelte Meldung’** eingetragen.

Der **‘Projekt’**-, **‘Tätigkeit’**- und **‘Faktura’**-schlüssel wurde in die entsprechenden Felder eingetragen (beim Projektleiter zu erfragen).

Der Realaufwand für die Prüfung wurde in dem Feld **‘Real Gesamt’** eingetragen.

Dokumente, die im Rahmen der Prüfung erstellt wurden, sind mit der Meldung verknüpft worden.

Da die Meldung erledigt ist, muß der Kunde nur noch eine Abnahmeerklärung schicken, mit der die Meldung dann geschlossen werden kann. Der Kundenbetreuer sendet die Meldung via Posteingangskorb an den Projektkoordinator, der für die Prüfung und Verfolgung der Abnahmeerklärungen zuständig ist. Danach kann die Meldung aus dem Posteingangskorb des Kundenbetreuers entfernt werden.

=> Ende

Schritt 3:

Der Entwickler nimmt die Meldung entgegen (übernimmt die Verantwortung), wenn er das Feld **‘Bearbeitungsstatus’** auf **‘Entwickler fixed’** setzt.

Vorgabe für den Entwickler sind die unter der Meldungsnotiz **‘Wie wird der Fehler reproduziert’**, **‘Umgehung des Fehlers’** und **‘Lösungsansatz’** eingetragenen Informationen. Ausführliche Konzepte können zusätzlich über separate Dokumente mit der Meldung verknüpft werden.

Der Entwickler verfährt bei der Bearbeitung der Meldung wir im Abschnitt Entwicklungszyklus beschrieben[33].

Nachdem der Entwickler die veränderten Dateien zu der Meldung wieder in PVCS Version Manager eingecheckt hat, dokumentiert er für jede Datei die Verknüpfung zur Meldung in PVCS-Tracker. Dort werden der Dateiname, Check-out-Revsion, Check-In-Revision und Change Text (der gleiche Text wie in PVCS Version Manager) erfaßt.

Änderungen am Konzept werden in Absprache mit dem Kundenbetreuer unter der Meldungsnotiz **‘Lösungsansatz’** dokumentiert. Gleiches gilt für die Meldungsnotiz **‘Wie wird der Fehler reproduziert’**, **‘Umgehung des Fehlers’** und **‘Testanleitung’**.

Der Realaufwand für die Entwicklung und ggf. die Dokumentation (wenn vom Entwickler verfaßt), wird in den entsprechenden Feldern eingetragen (**‘Real Entwicklung’, ‘Real**

[33] siehe Abschnitt 5.2.5

Dokumentation'). Ergänzungen an der Konzeption werden ebenfalls eingetragen (Feld **'Real Konzept'**).

Der Entwickler reicht die Meldung an den Kundenbetreuer weiter, indem er in dem Feld **'Meldung gesendet an'** den Kundenbetreuer einträgt und ihm die Meldung via Posteingangskorb zusendet. Danach kann die Meldung aus dem Posteingangskorb des Entwicklers entfernt werden.

=> weiter mit Schritt 4

Schritt 4:

Der Kundenbetreuer nimmt die Meldung wieder entgegen (übernimmt die Verantwortung), wenn er den **'Bearbeitungsstatus'** auf **'Kundenbetreuer testet'** setzt.

Wenn beim Test Fehler entdeckt werden, geht die Meldung zurück an den Entwickler (**'Meldung gesendet an'**). Der Kundenbetreuer dokumentiert die Fehlerbeschreibung unter der Meldungsnotiz **'Testergebnis'**. Den bisherigen Testaufwand trägt er unter **'Real Test'** ein.

Wenn der Test Fehlerfrei verlaufen ist, trägt der Kundenbetreuer ebenfalls den Testaufwand ein. Zusätzlich errechnet er den tatsächlichen Gesamtaufwand für die Bearbeitung der Meldung **('Real gesamt'**) und ergänzt unter der Meldungsnotiz **'Auslieferungstext'** den Auslieferungstext. Die Dokumentation wird abschließend geprüft und der Aufwand wird unter **'Real Dokumentation'** zusammengefaßt. Anschließend sendet der Kundenbetreuer die Meldung an den Versionsverwalter, der für die Integration[34] in ein Programmrelease verantwortlich ist.

4.2.2 Änderungsmanagement mit PVCS Tracker

Der PVCS Tracker von Merrant ist Änderungsmanagement- und Fehlerverfolgungssystem. Es steht in seiner derzeitigen Version nur auf der Windowsplattform und mit eingeschränkter Funktionalität auf Browserbasis zur Verfügung. Der Kern von Tracker ist eine zentrale Datenbank, in der alle Änderungsanforderungen bzw. Fehlermeldungen in strukturiert und in regelbasiert gespeichert werden. Während des Lebensdauer solcher Meldungen greifen unterschiedliche Projektteilnehmer auf diese Meldungen zurück und dokumentieren ihre jeweilige Arbeit an den Meldungen im System. Dies schafft eine umfassende Datenbasis über Aktivitäten im Entwicklungs- bzw. Änderungsprozeß, die

[34] siehe Abschnitt 5.2.6

detaillierte Informationen über den jeweils aktuellen Projektstatus liefert. Da jedes Projekt unterschiedliche Anforderungen an ein Änderungsmanagement stellt, bietet Tracker ein generisches Datenmodell, in dem je nach Anforderung zusätzliche Felder ergänzt werden können. Somit lassen sich individuelle Änderungsmeldungen entwerfen und pflegen. Die Datenbasis kann über ein Reporting-Werkzeug ausgewertet werden. Es stehen vordefinierte Reports wie z.B. Trendanalysen zur Verfügung. Die Integration mit den anderen PVCS Produkten ist erst in den neueren Versionen ausgereift. Dies liegt daran, daß Tracker als Fremdprodukt hinzugekauft wurde. Das Berechtigungskonzept ist rollenbasiert und sehr fein einstellbar. Es können z.B. nicht nur Rechte mit Datenfelder verknüpft werden, sondern auch mit Ereignissen verbunden werden. Damit läßt sich eine rudimentäre Prozeßunterstützung realisieren. Eine explizite Prozeßdefinition und –unterstützung ist in dem Tracker jedoch nicht vorgesehen. Auf der Datenbankseite können handelsübliche Datenbanksysteme (bei PRORIS Sybase) zum Einsatz kommen. Dies gewährleistet Skalierbarkeit, Zuverlässigkeit und Sicherheit des Systems. Auch die Datensicherungsmechanismen entsprechen damit dem im Abschnitt 3.5 beschriebenen Standard.

4.3 Bewertung des Änderungsmanagement

Zum Anfang ist zu bemerken, daß allein die Tatsache, daß ein kommerzielles SKM-System für das Änderungsmanagement zum Einsatz kommt, positiv zu bewerten. Laut der Analystenfirma Ovum arbeiten derzeit noch 85 Prozent aller Softwareprojekte mit keinen oder nicht von Herstellern unterstützten SKM-Werkzeugen[IT Fokus 7/2000]. Durch den Einsatz von PVCS-Tracker ist es gelungen, alle Meldungen zu dokumentieren und steuerbar zu machen. Bei der vorherigen manuellen Bearbeitung, gab es keine einheitlichen Strukturen. Oft war nicht klar, ob eine Meldung schon aufgenommen war oder sogar schon bearbeitet wurde. Der Änderungsprozeß ist somit transparenter und besser steuerbar. Fehleranfällige Systemteile können identifiziert und ggf. erneuert werden. Die Dokumentation zu einem Vorgang (bzw. einer Meldung), findet sich nun an einem zentralen Punkt; der PRORIS-Meldung in PVCS-Tracker. Damit ist die Organisation der Projektdokumentation verbessert worden. Ein weiter Pluspunkt ist, daß Releasenotes automatisch erzeugt werden können.

Es gibt jedoch auch eine Menge Schwachpunkte des Systems und somit Verbesserungsmöglichkeiten. Es gibt nur wenig Prozeßunterstützung im PVCS-Tracker. Regeln, Abhängigkeiten und Reihenfolgen können nur über aufwendige Ablaufbeschreibungen[35] dargestellt werden. Der PVCS-Tracker unterstützt dies nur mit

[35] siehe Abschnitt 4.2.1

seinem flexiblen Datenmodell, in dem solche Steuerfelder definiert werden können. Die Regeln können vom System aber nicht geprüft werden und müssen somit extern dokumentiert werden. Die Anwender müssen dann das System nach diesen Regeln bedienen, damit der gewünschte Effekt eintritt. Dieses System ist fehleranfällig und wird von einigen Projektmitarbeitern umgangen. Das kann wiederum zu verfälschten Ergebnissen führen. Ein Prozeßunterstützung würde zu einer erheblich höheren Akzeptanz des Systems führen, da die Regeln dann im System abgelegt sind, und nicht mehr jedes Teammitglied sie im Kopf haben muß.

Ein weiter Schwachpunkt ist die unzureichende Integration mit dem Versionsmanagement. Eine Verbindung zwischen PRORIS-Meldung und den Änderungen an den betroffenen Programmdateien, kann nur umständlich hergestellt werden. Der Änderungstext muß erneut im PVCS-Tracker eingetragen werden, obwohl er im Versionsmanagement bereits erfaßt wurde. Dieser unzureichende Komfort und die Redundanz bei der Eingabe ist ein weiterer Punkt, der zu mangelnder Akzeptanz des System bei den Entwicklern führt. Der Verwaltungsaufwand für eine Quellcodeänderung wird sehr hoch.

Des weiteren werden keine Abhängigkeiten zwischen Meldungen oder eine Meldungshierarchie unterstützt. Dies wäre ein nützliches Strukturierungselement um z.B. auch eine bessere Verbindung zu Projektmanagement-Werkzeugen zu ermöglichen.

Bei Fehlermeldungen müssen häufig Bildschirmausdrucke gemacht werden, um die Fehlersituation zu beschreiben. Solche Bildinformationen können nicht mit einer Meldung komfortabel verknüpft und auch nicht betrachtet werden. Dieses führt zu einer redundanten manuellen Aktenführung für PRORIS-Meldungen mit Bildinformationen.

Der Prozeß der Eingabe aller PRORIS-Meldungen ist für das PRORIS-Team relativ aufwendig. Auch der Austausch von Meldungslisten und Statusreports mit den Kunden ist ein aufwendiger Prozeß. Daher wäre es sinnvoll, wenn die Kunden die Möglichkeit bekämen, vom Internet aus neue Meldungen einzustellen und Reports und Statusinformationen abzurufen. Aufgrund von OS/2 muß derzeit noch eine alte PVCS-Tracker Version zum Einsatz kommen, bei der diese Möglichkeit nicht gegeben ist.

Das Änderungsmanagement ist für ein Projekt von der Größe von PRORIS relativ weit fortgeschritten. Der Prozeß ist identifiziert und spezifiziert worden. Und das Werkzeug PVCS-Tracker wird im Rahmen seiner Möglichkeiten zur Unterstützung des Prozesses verwendet. Akzeptanzprobleme müssen durch einen schlankeren Prozeß und bessere Werkzeugintegration abgebaut werden.

5 Aufbau des PRORIS Versionsmanagement

Grundlage für das Versionsmanagement ist die Konfigurationsidentifizierung. Dabei geht es darum, die für eine Konfiguration relevanten Elemente zu bestimmen sowie notwendige Numerierungsregeln und Namenskonventionen aufzustellen.

5.1 Konfigurationsidentifizierung in PRORIS

5.1.1 PRORIS-Produktstruktur

Das PRORIS-System besteht aus diversen PRORIS-Anwendungen, die jeweils als eigener Prozeß auf dem Arbeitsplatzrechner ablaufen. Die PRORIS-Systemverwaltung spielt in diesem Zusammenhang eine herausgehobene Rolle, da nur über sie die Anmeldung am PRORIS-System durchgeführt werden kann. Die Systemverwaltung regelt somit den sicheren Zugang, ruft bei Bedarf die weiteren Anwendungen auf und steuert auch diese.

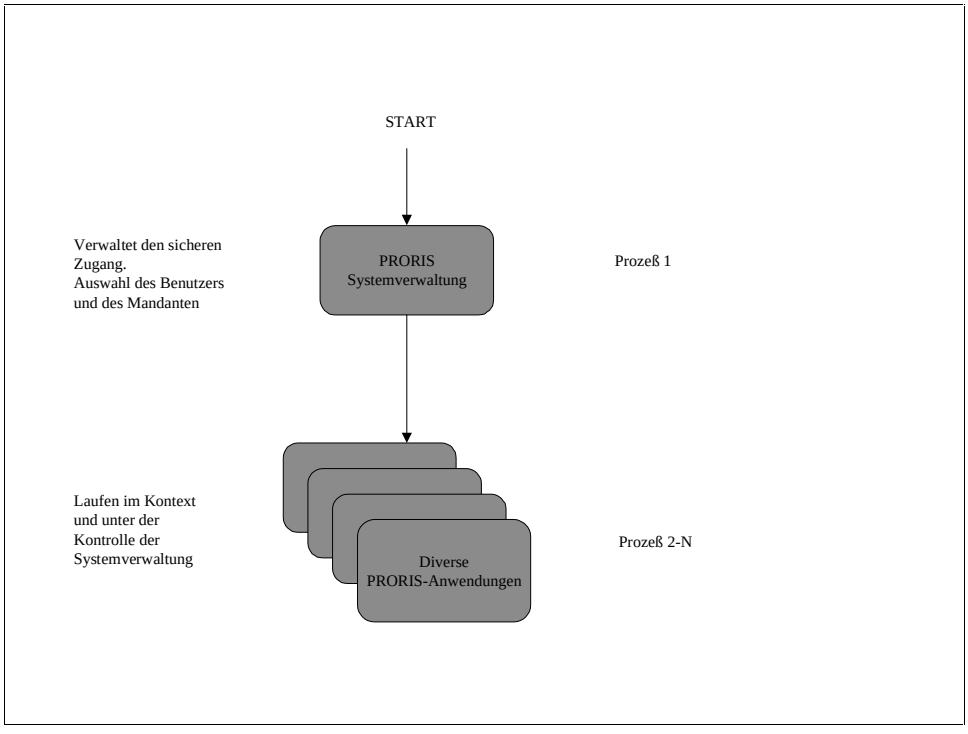


Abb. 5-1: Produktstruktur

Aus dieser Produktstruktur leitet sich auch die Modulstruktur ab. Jede PRORIS-Anwendung besteht aus einem ausführbaren Programm (EXE) und einigen dynamischen

Bibliotheken (DLLs), die Funktionalität für dieses Programm beinhalten. Hinzu kommen weitere DLLs, die übergreifender Natur sind und von mehreren oder sogar allen PRORIS-Anwendungen benutzt werden. Zur Zeit gibt es 24 PRORIS-Anwendungsmodule. Die Anwendungsmodule gehorchen einer einheitlichen Namenskonvention, die eine einfache und schnelle Zuordnung eines KE erlaubt.

Allen Anwendungsdateien wird hierzu ein zweistelliger Präfix vorangestellt, der die jeweilige Anwendung identifiziert. An der dritten Stelle folgt ein Kennzeichen, das darüber Aufschluß gibt, um welche Art von Modul bzw. Datei es sich handelt. Dabei wird zwischen folgenden Arten unterschieden:

P Module zur Präsentation der Anwendung

L Module für die Verarbeitungslogik

D Module für die Datenbeschaffung und –speicherung

R Module für das Reporting

Die restlichen fünf Stellen können frei vergeben werden. Die EXE und DLL beschränken sich auf die drei notwendigen Stellen, während die Dateien mit dem Programmcode alle acht Stellen ausnutzen. Hier noch einmal die allgemeine Form der Namenskonvention:

MMA?????.SSS

Wobei „MM“ für Modulpräfix, „A“ für Art des Moduls, „?“ für beliebig und SSS für den Dateityp[36] steht. Dieses Format leitet sich aus den 1990 noch gültigen Dateinamenskonventionen für OS/2 ab. Zum damaligen Zeitpunkt waren längere und damit aussagekräftigere Dateinamen nicht möglich.

[36] Für Dateitypen siehe auch Abschnitt 5.1.4

Es folgt ein Beispiel für die Systemverwaltung von PRORIS, die den Präfix „SV“ hat:

SVP.EXE		Das Programm Systemverwaltung
	Svpmain.c	Datei mit dem Hauptprogramm der Systemverwaltung
SVL.DLL		Die Logik-DLL der Systemverwaltung
	Svlbensu.c	Datei mit der Programmlogik für den „Benutzer suchen“-Dialog
SVD.DLL		Die Datenbank-DLL der Systemverwaltung
	Svdbensu.c	Datei mit der Datenbanklogik für den „Benutzer suchen“-Dialog
SVR.DLL		Die Report-DLL der Systemverwaltung
	Svrbenli.c	Datei mit dem Benutzer-Report

Die anwendungsübergreifenden Module haben individuelle Aufgaben und gehorchen deswegen keiner einheitlichen Namenskonvention.

Die Verzeichnisstruktur von PRORIS leitet sich aus der Modulstruktur und deren Namenkonventionen ab. Für die Gesellschaftsverwaltung (Präfix „GS“) ergibt sich somit folgende Verzeichnisstruktur:

\GSP	Wurzelverzeichnis der Gesellschaftsverwaltung (GSP.EXE)
\GSP\SRC	Hauptmodul von GSP
\GSP\INC	Headerdateien die nur in GSP verwendet werden
\GSP\RES	Dialogdefinitionen, Texte, Icons für GSP
\GSP\HELP	Hilfedateien für GSP
\GSP\LOGIC\	Wurzelverzeichnis der Logik DLL (GSL.DLL)
\GSP\LOGIC\SRC	Programmlogik für die Dialoge der Gesellschaftsverwaltung
\GSP\DB\	Wurzelverzeichnis der Datenbank DLL (GSD.DLL)
\GSP\DB\SRC	Programmlogik für die Datenbankzugriffe
\GSP\REPORTS\	Wurzelverzeichnis der Report DLL (GSR.DLL)
\GSP\REPORTS\SRC	Programmlogik für die Reports der Gesellschaftsverwaltung
\GSP\REPORTS\REP	Programmlogik für die Reports der Gesellschaftsverwaltung

Alle anderen Anwendungsmodule folgen demselben Aufbau. Die übergreifenden Module haben meistens keine so tiefe Hierarchie der Verzeichnisse, da sie häufig nur aus einer DLL bestehen. Sie folgen aber grundsätzlich denselben Namenskonventionen. Die Verzeichnisstruktur für PMAPI.DLL sieht z.B. folgendermaßen aus:

\PMAPI	Wurzelverzeichnis der PMAPI (PMAPI.DLL)
\PMAPI \SRC	Programmlogik von PMAPI
\PMAPI \INC	Headerdateien die nur in PMAPI verwendet werden
\PMAPI \HELP	Hilfedateien für PMAPI

5.1.2 PRORIS-Modulübersicht

Es folgt eine kurze Beschreibung aller PRORIS-Module:

PMAPI

Dies ist die zentrale DLL von PRORIS, die alle Systemfunktionen für die Präsentationsschicht zur Verfügung stellt. Hier werden alle Betriebssystemabhängigkeiten gekapselt. Dies erfolgt durch Variantenbildung mit Hilfe von bedingter Kompilierung[37].

DB_NET bis DB_NET7

Für jede Tabelle der PRORIS-Datenbank existiert eine Datei mit Extension SQC. Diese Dateien enthalten jeweils alle SQL-Befehle, die Zugriffe auf die betreffende Tabelle durchführen. Alle Zugriffe auf die Datenbank erfolgen über diese Module. Die Aufteilung der SQC-Dateien auf mehrere DLLs ist durch die Größe der Module bedingt. Diese Kapselung aller SQL-Befehle ermöglicht einen leichteren Wechsel zu einem anderen Datenbanksystem.

DB_BIND

Diese Datei ist eine Hilfs-DLL, die Datenbankthreads[38] startet und Datenbank-DLLs sowie Reports-DLLs von PRORIS-Anwendungen bei Bedarf nachlädt, damit nicht immer alle DLLs im Hauptspeicher gehalten werden müssen. Es werden die ??D.DLLs und ??R.DLLs von Proris nachgeladen.

DBAPI

Diese DLL enthält häufig gebrauchte Datenbank-Funktionen von PRORIS. Die DLL stellt einen Container dar, in dem Funktionen, die etwas mit der Datenbank zu tun haben und die von mehreren Anwendungen benutzt werden, gesammelt werden. Zusätzlich sind Funktionen zur Transaktionssteuerung und Fehlerbehandlung für die Datenbank enthalten.

[37] siehe Abschnitt 5.1.6
[38] Threads: Unabhängig ablauffähige Codefragmente die vom Betriebssystem eigene Zeitscheiben zugeordnet bekommen

SEIFLB32

Diese DLL stellt eine mehrspaltige Listbox zur Verfügung. Unter OS/2 ist diese DLL ein Fremdprodukt der Firma SES, unter NT eine Eigenentwicklung (EZTW32.DLL), die von der Control-Klasse SysListView32 abgeleitet ist.

SUBCLASS

Diese DLL ist ebenfalls eine Eigenentwicklung ist stellt formatierte Eingabefelder, Ausgabefelder und Comboboxen zur Verfügung, insbesondere Datums-, Zeitfelder und weitere formatierte Ein- und Ausgabefelder. Definiert werden die Control-Klassen WC_UDENTRY, WC_UDSTATIC, WC_UDCOMBO, WC_UDBUTTON.

GLOBSRC

Hier finden sich die Standard-Fensterprozedur und Standard-Dialogprozedur von Proris. Alle relevanten Ereignisse von Fenstern, Dialogen und des Betriebssystems werden hier abgefangen, bevor sie an die Dialoge weitergeleitet werden.

SYMSGRES, SYMSGEN

Dieses sind zwei reine Resouce-DLLs, die keinen Programmcode enthalten. In symsgres.dll befinden sich alle Meldungstexte von PRORIS, u.a. Meldungstexte für Messageboxen. Symsgen.dll ist die englischsprachige Version. In diesem Fall handelt es sich um um Variantenbildung, die durch Trennung in zwei DLLs bewirkt wurde. Abhängig davon welche Sprache vom Benutzer gewünscht ist, wird die deutsche oder englische DLL geladen.

PROREPLB

Diese DLL stellt Funktionen für Reports zur Verfügung. Diese werden von den Reports-DLLs (??R.DLL) gebraucht. Reports-DLLs enthalten Rep-Dateien, die von zwei im PRORIS-Team entwickelten Compilern übersetzt werden (prorep und sqlpp). Proreplb.dll stellt die Runtime-Bibliothek für diese Compiler dar.

??P.EXE, ??L.DLL, ??D.DLL, ??R.DLL

Jede PRORIS-Anwendung besteht aus einem Exe-File und 3 DLLs. ??L.DLL ist die sog. Logik-DLL, ??D.DLL die Datenbank-DLL und ??R.DLL die Report-DLL.

Hinzu kommen unter OS/2 noch die Hilfedateien ??P.INF und ??P.HLP, unter Windows NT sind dies die Dateien ??P.HLP und ??P.CNT (ebenfalls für die Online-Hilfe)

Die Resourcen einer Anwendung sind auf EXE und Logik-DLL aufgeteilt. Die Menüs und die Strings befinden sich im Resource-File der EXE, die Dialogresourcen befinden sich in der Logik-DLL (??L.DLL)

SYL, SYD, SYR

Diese drei DLLs enthalten Dialoge, die von mehreren Anwendungen benutzt werden, die sog. „Syboxes-Dialoge“. Die hier enthaltenen Dialog stehen allen PRORIS-Anwendungen zur Verfügung und sind zum Großteil Such- und Auswahldialoge.

NOV

Diese DLL dient der Integration von PRORIS in Novell-Netzwerke. Sie wird nur geladen, wenn auf dem Arbeitsplatz ein Novell-Client installiert wurde.

DB_HOST

Diese DLL enthält Zugriffsfunktionen auf Host-DB2-Tabellen. Diese werden für die Host-Schnittstelle von PRORIS benötigt.

DB_DDCS

Diese DLL enthält Funktionen für verteiltes Transaktionsmanagement unter PRORIS.

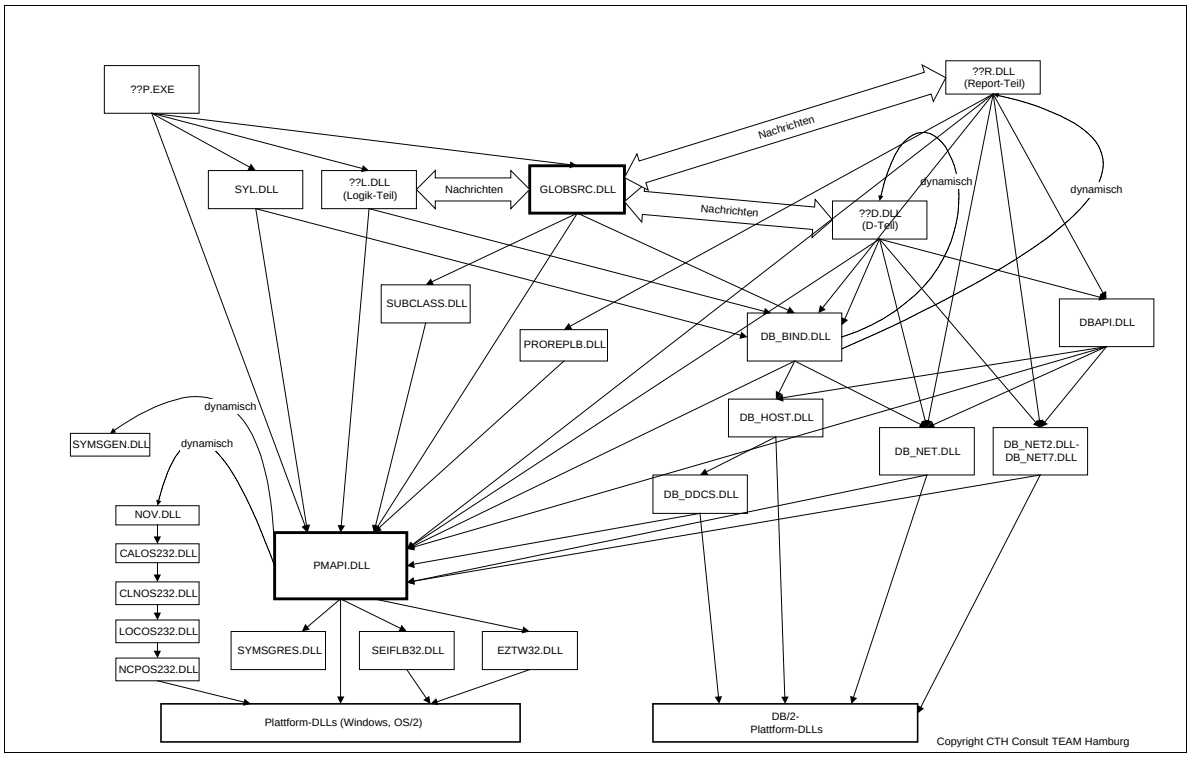


Abb. 5-2: PRORIS-Modulhierarchie [PRORIS 99]

Abbildung 5-2 beschreibt die Abhängigkeiten der PRORIS-Module untereinander sowie deren Beziehungen zu den Systemschnittstellen. Es wird ein komplexes Beziehungsgeflecht deutlich, das bei dem Buildmanagement[39] berücksichtigt werden muß, um die richtige Reihenfolge bei der Erstellung der Module einzuhalten. Zusätzlich zeigt diese Übersicht, welche Module erneut erstellt werden müssen, wenn sich ein bestimmtes Modul geändert hat.

5.1.3 Datenbankdefinitionen

Neben den Programmodulen unterliegen auch die Datenbankdefinitionen dem Versionsmanagement. Zu den Datenbankdefinitionen gehören alle Dateien, die für den Aufbau einer neuen Datenbank bzw. das Verändern einer bestehenden Datenbank notwendig sind. Dies gilt sowohl für die Struktur der Datenbank als auch für grundlegende Systemdaten die bei der Installation oder einem Releasewechsel ausgeliefert werden. Die Struktur einer relationalen Datenbank wird mit Hilfe der sogenannten Data Definition

[39] Siehe Abschnitt 5.3

Language (DDL) beschrieben. Die Systemdaten werden mit der Data Manipulation Language (DML) eingefügt bzw. verändert. Beide Definitionssprachen gehören zum Sprachumfang von SQL. Die Datenbankdateien werden in folgender Verzeichnisstruktur verwaltet:

\RDBDEF	Wurzelverzeichnis der Datenbankdefinitionen
\RDBDEF\DB2DEF	Datenbankdateien zum Aufbau einer neuen Datenbank
\RDBDEF\DBUPDATE	Datenbankdateien zum Verändern einer bestehenden Datenbank
\TABLE	Tabellendefinitionen
\VIEW	Viewdefinitionen
\REF	Referentielle Integritäten
\INDEX	Indices
\SYSDATA	Systemdaten
\USRDATA	Kundendaten
\USRDATA\<KUNDE>	pro Kunde
\EINMAL	Einmalige Datenbankveränderungen
\EINMAL\<KUNDE>	pro Kunde

In dem Verzeichnis „DB2DEF“ wird die Datenbankstruktur verwaltet. Es gibt jeweils eine Datei für die Tabellendefinitionen, Viewdefinitionen, referentiellen Integritäten, die Indices und die Kommentare in der Datenbank. Diese Dateien stellen den jeweiligen Status Quo der Datenbankdefinitionen dar und werden zum Aufbau einer neuen, nicht mit Daten gefüllten, Datenbank verwendet.

Unter dem Verzeichnis „DBUPDATE“ werden die laufenden Veränderungen an den Datenbankdefinitionen verwaltet. Dies gilt für die Datenbankstruktur sowie die System- und Kundendaten. Die einmaligen Datenbankaktionen und die Kundendaten werden pro Kunde unterschieden.

Bei der Verwaltung von Datenbankänderungen auf Tabellen muß mit größter Vorsicht verfahren werden, denn sie können nicht einfach wie andere Definitionen ersetzt werden. Bei Tabellenänderungen müssen die bestehenden Daten vor der Änderung gesichert und anschließend in die neue Struktur überführt werden. Bei mehreren Änderungen auf einer Tabelle müssen somit die entsprechenden Veränderungen in chronologischer Reihenfolge durchgeführt werden.

Für die Dateien unter dem Verzeichnis „DBUPDATE“ gelten folgende Namenskonventionen:

EINMAL:	EINMAL??.DAT
INDEX:	<Tabellenname>_<Index-Name>.IDX
REF:	<Tabellenname>_<Ref. Int. Name>.REF

SYSDATA:	<Tabellenname>.DAT
TABLE:	<Tabellenname>.DDL, UPD, SAV
USRDATA:	<Tabellenname>.DAT
VIEW:	<Viewname>.DDL

Im Gegensatz zu den Namenskonventionen für Programmdateien, werden hier lange Dateinamen verwendet. Dadurch lassen sich die benötigten Dateien sehr einfach auffinden.

5.1.4 Typen von verwalteten KE

In PRORIS werden unterschiedliche Dateitypen der Versionsverwaltung unterworfen. Dies sind unter anderem die in Abschnitt 5.1.1 beschriebenen PRORIS-Programmdateien und PRORIS-Hilfedateien. Dabei werden jedoch nur die Quellelemente verwaltet, da alle abgeleiteten Dateien jederzeit wieder erzeugt werden können. Eine Ausnahme bilden hier die Dateien, die an den Kunden ausgeliefert werden (ausführbare Programmdateien, Bibliotheken und Hilfedateien). Sie werden zwar nicht im Versionsverwaltungssystem abgelegt, unterliegen aber einer separaten Verwaltung in den Kundenumgebungen.

Weitere Dateien, die der Versionverwaltung unterliegen, sind die MS Word Vorlagen, die oben genannten Datenbankdefinitionen, Hilfsprogramme und –kommandos für die Entwicklungsumgebung sowie Dateien mit Parametern für die Entwicklungsumgebung.

Zur Zeit nicht unter der Kontrolle der Versionsverwaltung sind die Compiler der Fremdhersteller, Spezifikationen im Word-Format, Betriebssystem-Bibliotheken sowie Testdaten und Testfälle. Es folgt eine Liste der im Versionsmanagementsystem verwalteten Dateitypen:

Programmdateien

*.C	C-Sourcen
*.DEF	Definition-Files (Linker Informationen)
*.MAK	Make-Files (Beschreibt wie ein Modul generiert wird)
*.STS	Status-Files (Speichert die Einstellungen der Workbench)
*.PCC	Dateien mit einem erweiterten SQL-Dialekt (Eigener Präcompiler)
*.IPF	Hilfe-Dateien (Text und Layout)
*.IPM	Master-Hilfe-Dateien (Text und Layout)
*.H	C-Header
*.RCH	Ressource-Header
*.REP	Reportsdefinitionen (Eigener Präkompiler)
*.DLG	Dialogdefinitionen
*.RC	Ressource-Files (Menüdefinitionen, Texte, Hilfedefinitionen)
*.BND	Binde-Pläne (SQL-Zugriffspläne)
*.SQC	Datenbankzugriffe (Embedded SQL, dynamisches SQL)
*.BMP	Bitmaps

*.DOT	MS Word Vorlagen
*.DOC	MS Word Dokumente
*.CMD	Kommandodateien
*.CBL	Cobol-Sourcen
*.ICO	Icons (Bildsymbole)
*.CUR	Cursor-Definitionen (Grafiken)

Datenbankdateien

*.DDL	Definitionen von Tabellen und Views
*.SAV	Sicherung von bestehenden Tabellendaten
*.UPD	Aktualisierung von Tabellen
*.REF	Definition von referentiellen Integritäten
*.IDX	Definitionen von Indices
*.DAT	Systemdaten, Kundendaten, einmalige Datenbankänderungen

5.1.5 Numerierungsschema für das Versionsmanagement

Bei den Revisionen wird das Numerierungsschema des eingesetzten Versionsmanagementsystems - PVCS Version Manager (VM) - verwendet. Dies sieht eine einfache Numerierung beginnend mit 1.0, 1.1, 1.2 usw. vor. Temporäre Varianten (Zweige) werden durch das Einfügen einer weiteren Numerierungsebene abgebildet. Ein Zweig auf der Revision 1.2 führt somit zu der Revsionsnummer 1.2.1.0. Weitere Änderungen in diesem Zweig werden entsprechend fortgeschrieben (1.2.1.1, 1.2.1.2, etc.). Ein zweiter Zweig auf der Revision 1.2 führt zu der Revisionsnummer 1.2.2.0. Bei Änderungen in diesem Zweig wird wie zuvor mit 1.2.2.1, 1.2.2.2, usw. fortgefahren. Langfristige Varianten werden unter 5.1.6 behandelt.

Das Definieren von Bezugskonfigurationen, wird über sogenannte Versionslabel abgebildet. Einer Datei, in einer bestimmten Revision, können beliebig viele Versionslabel zugeordnet werden. Somit lassen sich beliebige Konfigurationen von Dateien in bestimmten Revisionen erstellen. Der Aufbau des Versionslabels unterliegt in PRORIS folgender Namenskonvention.

Vollversion (Komplette Auslieferung)

„V“RRVVVPP.KKK

Patch (Teilauslieferung)

„P“RRVVVPP.KKK

Mit

RR als zweistellige Releasenummer

VVV als dreistellige Versionsnummer

PP als zweistellige Patchnummer

und

KKK als dreistellige Kundennummer.

Das Präfix „V“ bzw. „P“ dient lediglich der besseren Lesbarkeit von Versionslabeln und es gilt folgende Regel:

wenn Präfix = „V“ dann PP = 00 und

wenn Präfix = „P“ dann PP<> 00.

Die dreistellige Kundennummer trägt der Tatsache Rechnung, daß z.Zt. prinzipiell für jeden Kunden eine eigene Patchhistorie auf seiner installierten Vollversion geführt wird.

Diese Namenskonvention wird auch für die Auslieferungsverzeichnisse in den Kundenumgebungen bei der CTH und in Test und Produktion bei den Kunden verwendet. Dieser Label dient somit als allgemeiner Identifikator einer PRORIS-Auslieferung bzw. einer PRORIS-Bezugskonfiguration.

5.1.6 Langfristige Varianten

Die OS/2- und die Windows NT-Version von PRORIS basieren auf demselben Sourcecode. Die spezifischen Betriebssystemschnittstellen von OS/2 und Windows NT werden von einer betriebssystemspezifischen Schicht in PRORIS gekapselt. Die entsprechenden Unterschiede sind über bedingte Kompilierung sowie schnittstellenabstrahierende Makros implementiert. Damit wird zur Zeit der Kompilierung nur der jeweils plattformspezifische Programmode sowie die plattformspezifischen Headerdateien berücksichtigt. Jegliche Anwendungslogik ist somit unabhängig von den beiden Betriebssystemen. Für beschreibenden Dateien, in denen z.B. Dialogdefinitionen, Hilfetexte oder Fehler- und Meldungstexte abgelegt sind, wurden Migrationswerkzeuge entwickelt, die eine automatische Überführung vom OS/2- in das Windows NT-Format ermöglichen. Hierbei müssen z.B. die unterschiedlichen Koordinatensysteme und Zeichensätze in OS/2 und Windows NT berücksichtigt werden. Diese Umwandlung von OS/2 in Windows NT muß nicht explizit vorgenommen werden. Die Werkzeuge und

Einstellungen sind in die Entwicklungsumgebung integriert und kommen automatisch während des Übersetzungsprozesses zur Anwendung. Es werden z.B. die Dialogbeschreibungen (Ressouredateien), bevor sie mit dem Windows NT Ressourcecompiler übersetzt werden, vom OS/2- in Windows NT Format umgewandelt. Für den Anwendungsentwickler stellt sich diese Umwandlung als integrierter Übersetzungsprozeß dar. Die OS/2- und die Windows NT Version stellen somit zwei Varianten ansonsten gleicher Funktionalität dar. Die Variantenbildung ist hierbei nicht über unterschiedliche Dateien, sondern über bedingte Kompilierung innerhalb derselben Dateien bzw. automatische Formatumwandlung gelöst worden.

Eine andere Technik der Variantenbildung wird bei der Forderung nach kundenspezifischer Funktionalität angewendet. Dabei soll sich das Programm an bestimmten Stellen abhängig vom Kunden unterschiedlich verhalten. Dies wurde über die sogenannte PRORIS-Initialisierungsdatei realisiert[40]. In dieser Datei kann, neben vielen anderen Einstellungen, der Kunde eingestellt werden. Im Programmcode wird an den spezifischen Stellen nach dieser Kundeneinstellung abgefragt und zum entsprechenden Programmcode verzweigt. Der Vorteil ist wiederum, daß es nur eine Datei mit sämtlichen Varianten gibt. Somit müssen nicht verschiedene Programmdateien an die unterschiedlichen Kunden ausgeliefert werden, denn jedes PRORIS-Release enthält alle Kundenvarianten. Dieses Verfahren wird auch angewendet, um Unterschiede, die durch den Einsatz verschiedener Datenbankversionen entstehen, auszugleichen.

[40] Siehe Abb. 5-3

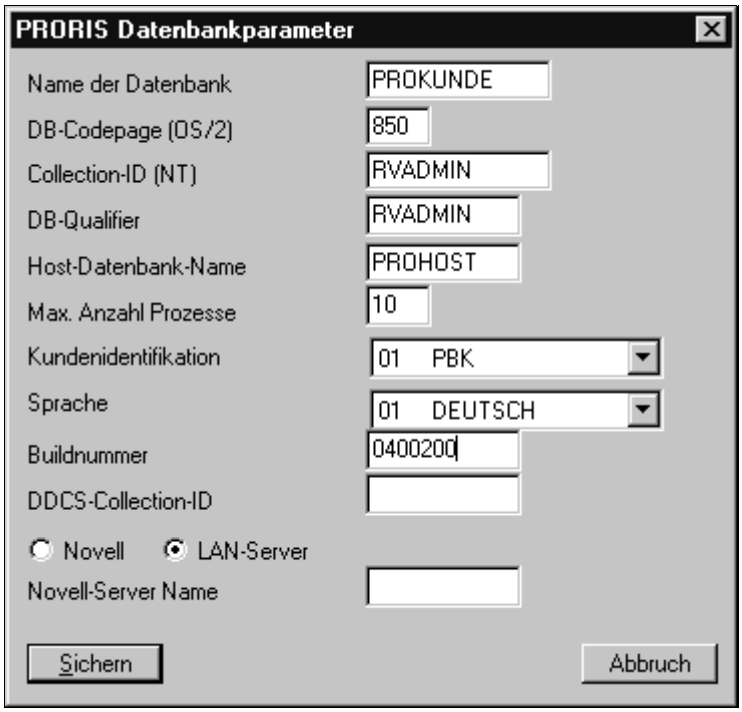


Abb. 5-3: PRORIS Initialisierungsdatei

Ein drittes Verfahren der Variantenbildung wird für Funktion, die nur bestimmte Kunden benötigen, angewendet. Hierbei wurde das PRORIS-Berechtigungssystem dazu verwendet, bestimmte Funktionalität den Kunden nicht zur Verfügung zu stellen. Dies ist sinnvoll, um eine gestaffelte Preisliste nach Funktionsbausteinen zu ermöglichen, ohne wiederum unterschiedliche Programmdateien ausliefern zu müssen. Das PRORIS-Berechtigungssystem unterscheidet dabei drei Berechtigungsebenen:

Die Anwendungsebene, die Menüebene und die Dialogobjektebene. Auf der Anwendungsebene können Berechtigungen für komplette PRORIS-Teilkomponenten vergeben werden. Berechtigungen für die Menüpunkte der Teilkomponenten werden auf der Menüebene eingestellt. Auf der Dialogobjektebene werden Berechtigungen auf der Feldebene vorgenommen. Somit kann, unabhängig von den vom Kunden erworbenen Lizenzen, der komplette PRORIS-Funktionsumfang ausgeliefert werden. Die lizenzrelevanten Einschränkungen müssen somit lediglich über das Berechtigungssystem konfiguriert werden.

5.2 Versionsverwaltung mit PVCS Version Manager

5.2.1 Allgemeines

Die oberste Organisations-Struktur innerhalb des PVCS Versions-Managers (VM) ist das *PROJEKT*. Mit dem Anlegen eines Projektes wird automatisch ein *Projekt-Folder* (Ordner im VM) angelegt, in dem alle Projektdateien (Objekte) hinterlegt sind. Zusätzlich werden in einer Projekt-Konfigurationsdatei sämtliche Projekt-Parameter global verwaltet. Beim Aufruf des VM wird zunächst das sogenannte *Master-Projekt* geöffnet, in dem projektübergreifende Parameter hinterlegt sind, die zur Anwendung kommen, wenn mehrere Projekte verwaltet werden sollen. Im PRORIS-Projekt sind die projektspezifischen Parameter hinterlegt.

5.2.2 Verzeichnisstruktur

Die PVCS-Entwicklungsumgebung beinhaltet ein Archiv-, ein Referenz-, ein Workfile- und ein Objekt-Verzeichnis.

Referenz-Verzeichnis (X:\PRORIS)*

In dem Referenz-Verzeichnis speichert der VM automatisch eine Kopie jeder Datei, die über die VM-Aktion „Check-In“[41] archiviert wird. Es beinhaltet somit die jeweils aktuellste Revision jeder Datei im schreibgeschützten Modus. Sinn des Referenz-Verzeichnisses ist es, daß Dateien zum inspizieren bzw. Drucken nicht explizit aus dem Archiv entnommen werden müssen. Die abgeleiteten Dateien (Objekt- und andere abgeleitete Dateien) werden auf dem Referenz-Verzeichnis nur dann aktualisiert, wenn eine Kunden-Version erstellt wird. Für die tägliche Entwicklung stehen die abgeleiteten Dateien in dem Objektverzeichnis X:\OBJECTS (siehe unten). In dem Referenz-Verzeichnis dürfen vom Entwickler keine Änderungen vorgenommen werden, außer implizit über die entsprechenden VM-Aktionen (Check In, Check Out)[42].

[41] siehe Abschnitt 5.2.5
[42] siehe Abschnitt 5.2.5

Archiv-Verzeichnis (X:\ARCHIVE)*

Das Archiv-Verzeichnis ist strukturgleich zum Referenz-Verzeichnis aufgebaut und beinhaltet ebenfalls sämtliche Dateien mit der dazugehörigen Entwicklungs-Historie. Mit jeder Änderung eines Entwicklers wird automatisch eine neue Revision[43] archiviert. Archive ermöglichen es, eine Revision einer Datei wiederherzustellen, indem eine Kopie der Datei mit den entsprechenden Deltas (Speicherung von Differenzen) sowie dem Entwicklerkennzeichen, dem Änderungsdatum und der Änderungsbeschreibung gespeichert werden. In dem Archiv-Verzeichnis dürfen vom Entwickler keine Änderungen vorgenommen werden, außer über die entsprechenden VM-Aktionen (Check In, Check Out).

Die Archivnamen unterscheiden sich von den entspechenden Original-Dateien lediglich durch die Extension:

*.c wird *.c_v
*.obj wird *.obv
*.exe wird *.exv
usw.

Dateien, deren Extension in den ersten beiden Stellen identisch sind (z.B. *.rc und *.rch), werden folgendermaßen archiviert:

*.rc wird *.rv_
*.rch wird *.rvh
*.dlg wird *.dvg
*.dll wird *.dvl
usw.

Workfile-Verzeichnis (U:\PRORIS)*

Jeder Entwickler hat ein eigenes Workfile-Verzeichnis, in dem er seine Entwicklungstätigkeiten durchführt. U:\PRORIS ist am Anfang ein leeres Verzeichnis ohne Dateien. Es existiert lediglich die PRORIS-Verzeichnisstruktur. In dieser Verzeichnisstruktur werden Arbeitskopien der Dateien eingestellt, wenn sie aus dem Archiv-Verzeichnis zur Bearbeitung entnommen werden. Nur innerhalb dieses Verzeichnisbaums dürfen Dateien bearbeitet werden. Wenn die geänderten Dateien über den VM wieder archiviert werden sollen, ist dies nur über das Workfile-Verzeichnis

[43] siehe Abschnitt 2.2.4

möglich. Nach der Archivierung werden alle Dateien, die sich im Workfile-Verzeichnis befinden, automatisch gelöscht. Die *.EXE-, *.DLL und *.LIB-Dateien werden automatisch in die Integrationsverzeichnisse generiert. Über diese Verzeichnisse werden dann auch die „lokalen“ Tests vorgenommen.

Objekt-Verzeichnis (X:\OBJECTS)*

Alle abgeleiteten Dateien (z.B. Objektdateien) liegen auf einem zentralen Objekt-Verzeichnis, in das alle abgeleiteten Dateien automatisch bei der Kompilierung eingestellt werden. Das bedeutet, daß bei der Kompilierung eines Entwicklers z.B. von ABPMAIN.C auf U:\PRORIS\ABP\SRC, die Objekt-Datei ABPMAIN.OBJ auf U:\OBJECTS\ABP\SRC gestellt wird. Auf X:\OBJECTS\BIN bzw. X:\OBJECTS\LIB stehen die jeweils aktuellsten Programm- und Library-Versionen für die PRORIS Entwicklungs-Umgebung. In diesem Verzeichnis wird eine Aktualisierung durchgeführt, wenn eine „lokale“ Entwicklung auf U:\PRORIS* abgeschlossen und getestet wurde.

5.2.3 Folder

Folder stellen eine logische Sicht auf die Projekt-Dateien dar. Mit Hilfe der Folder können Aktionen auf logischen Gruppen von Dateien ausgeführt werden. In den vordefinierten Foldern sind alle PRORIS-Unterverzeichnisse komplett abgebildet, wobei die Bezeichnung eines Folders dem entsprechenden Verzeichnisnamen in der PRORIS-Umgebung entspricht:

X:\PRORIS\ABP\	wird zu	Archive Abp
X:\PRORIS\ABP\LOGIC	wird zu	Archive Abp Logic
X:\PRORIS\ABP\LOGIC\SRC	wird zu	Archive Abp Logic Src

usw.

Zusätzlich existiert für jedes Modul ein Folder, in dem alle Dateien hinterlegt sind, die für eine Compilierung und das Binden benötigt werden (*.MAK, *.STS, *.DEF). Die Folder wurden angelegt, um die entsprechenden Dateien schneller auffinden zu können. So befinden sich beispielsweise in dem Folder „MAK_TMP“ die Dateien TMP.MAK,

TMP.STS, TMP.DEF, TMD.MAK, TMD.STS, TMD.DEF, TML.MAK, TML.STS und TML.DEF.

In den Foldern sind die Dateien aus den bekannten Unterverzeichnissen zu Einheiten zusammengefaßt und sämtliche Informationen hinterlegt, die von VM zur Handhabung durch die Entwickler benötigt werden. Alle Aktionen auf den Dateien und Archiven müssen über Folder erfolgen, da ansonsten die Gefahr von Inkonsistenzen mit dem Datei-System besteht.

Vom VM wird mit dem ersten "Lock" (exklusive Bearbeitung einer Datei) auf einer Datei ein Folder mit der Bezeichnung "Files Locked by <USER>" angelegt, in dem alle von dem jeweiligen Entwickler "gelockten", das heißt exklusiv belegten, Dateien enthalten sind.

“<USER>“ steht in diesem Zusammenhang für den angemeldeten Benutzer beim VM.

5.2.4 Folder, Versionslabel und Tracker-Meldungen

Aufgrund der unzureichenden Integration zwischen dem PVCS Version Manager und dem im Kapitel 4 beschriebenen PVCS Tracker, ist es schwierig auf Seiten des VM Änderungen an Quelldateien in Beziehung zu einer PRORIS-Meldung zu setzen. Für die Integration von Änderungen in bestehende Releasestände muß jedoch nachvollziehbar sein, welche Änderungen an welchen Konfigurationselementen aufgrund welcher Anforderung oder Fehlermeldung durchgeführt wurden. Um dies zu dokumentieren, wird für jede PRORIS-Meldung, die im Tracker angelegt wird, ebenfalls ein Folder im VM angelegt. Die Entwickler arbeiten, wenn sie eine Meldung bearbeiten, über diesen Folder und ordnen allen bearbeiteten KE diesem Folder zu. Zusätzlich bekommt die durch die Änderung erzeugte Revision des jeweiligen KE einen Versionslabel. Foldername und Versionlabel entsprechen den Schlüsselkriterien für eine PRORIS-Meldung. Sie setzen sich folgendermaßen zusammen:

Kundenkürzel/Meldungsdatum/Meldungsnummer/Nachtragsnummer

Beispiel XX/15.5.1999/1/1

Mit Hilfe dieser Regel ist es möglich, auch ohne direkte Unterstützung der Werkzeuge, eine Verbindung zwischen Änderungsmanagement und Versionsmanagement zu realisieren. Nachteil dieser Lösung ist der manuelle Pflegeaufwand, der fehleranfällig ist.

5.2.5 Der Entwicklungszyklus

Im Entwicklungszyklus werden alle Aufgaben erledigt, die mit der Änderungen von KE im Zusammenhang stehen. Ausgangspunkt ist immer eine PRORIS-Meldung, die entweder vom Kunden oder intern eingereicht wurde. Der Entwicklungszyklus findet sich in der Prozeßgrafik Abb. 4-2 in dem Punkt „Programmierauftrag“ wieder.

Entgegennahme des Änderungsauftrages im PVCS-Tracker

Die Änderungsaufträge werden ausschließlich über den PVCS-Tracker verwaltet. Jeder Entwickler kann sich im „In Tray“ seiner PVCS-Tracker Oberfläche die für ihn anstehenden Aufträge ansehen. Der „In Tray“entspricht einem elektronischen Briefkasten (Post-Eingangskorb), in den vom jeweiligen Fachbetreuer die Änderungsaufträge an die Entwickler „gesendet“ werden[44]. Damit sichtbar wird, daß der Entwickler die Bearbeitung übernommen hat, muß er den Status der entsprechenden Meldung von „Fachbetreuer prüft“ auf „Entwickler fixed“ setzen. Im folgenden werden die einzelnen Schritte im Entwicklungszyklus beschrieben.

a) „Check Out“ Make-, Status-, Definition-File (Icon-File)

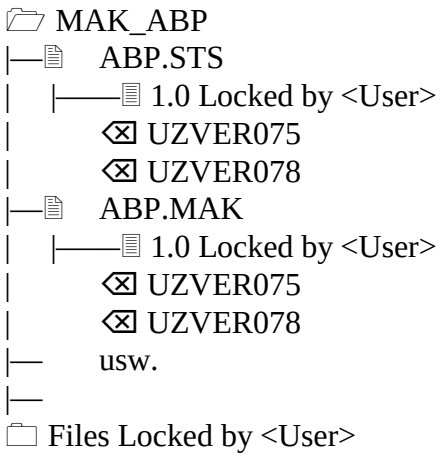
Die *.MAK, *.STS, und *.DEF- File (bei Bedarf auch das/die *.ICO-Dateien) der entsprechenden Anwendung werden exklusiv per „Check-Out“ aus dem Archiv entnommen. Zum schnelleren Auffinden der entsprechenden Dateien existiert für jede Anwendung ein Folder, in dem nur diese Dateien enthalten sind.

🗀 Doppelklick auf Folder = Open Folder

🗁 MAK_ABP
|— 🗎 ABP.STS
|— 🗎 ABP.MAK
|— 🗎 ABP.DEF
|— usw.

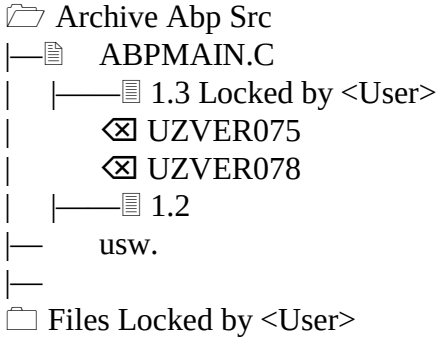
Die Dateien werden „gelockt“ und sind damit für andere Entwickler gesperrt. Zusätzlich wird automatisch ein Folder „Files Locked by <User> „ angelegt, in dem alle Dateien enthalten sind, die exklusiv entnommen wurden. Über diesen Folder erhält jeder Entwickler einen schnellen Überblick über die Dateien, die er in Bearbeitung hat.

[44] siehe Abb. 4-2



b) Check Out der zu ändernden Source-Dateien

Die zu ändernden Dateien werden per „Check-Out“ aus dem Archiv entnommen und sind damit für andere Entwickler gesperrt. Es werden nur die tatsächlich zu ändernden Sourcen entnommen. Alle anderen Dateien die zum Kompilieren oder Binden notwendig sind, werden automatisch aus dem Archiv gezogen. Sämtliche Objekt- und Zwischendateien werden somit nicht aus dem Archiv entnommen.(Bei Reports z.B. nur die *.rep Datei). Zum Beispiel:



In der folgenden Tabelle sind die Zusammenhänge zwischen Source-, Objekt- und Zwischendateien aufgeführt:

Quelle	Ziel	Zwischendatei(en)
*.c	*.obj	
*.rc	*.res	*.rcp
*.ipf	*.rtf	*.ip1

*.ipm	*.hlp	*.ip1 *.ip2 *.ip3
*.pcc	*.obj	*.pcu *.c
*.rep	*.obj	*.reo *.pc *.pco *.c
*.sqc	*.obj *.bnd	*.c

c) Setzen der Programm-Liste über die Workbench

In Workbench[45] wird das zu ändernde Modul aufgerufen. Der Aufbau der Make-Dateien entspricht den Konventionen des PVCS Configuration-Builder[46](CB). In den Make-Dateien werden die Quellelemente und die Regeln zur Übersetzung dieser Elemente hinterlegt.

Soll eine neue Datei in die Make-Datei aufgenommen werden, so müssen folgende Einträge vorgenommen werden:

```
# ***************************** Sourcen *****************************

CSRCS =     SourceNeu.c                 C-Sourcen

(entsprechen für andere Source-Dateien z.B. *.rc, ...)
```

Eine Besonderheit der CB Make-Dateien ist die sogenannte ScanDeps Region. ScanDeps.exe ist ein CB-Programm, das automatisch die Abhängigkeiten für die modulspezifischen Dateien generiert und in die Make-Datei zwischen den Schlüsselwörtern #UPDATE# und #ENDUPDATE# einfügt. Diese Liste der abhängigen Module muß vor jedem Übersetzungsprozeß aktualisiert werden.

d) Änderungen an den Sourcen vornehmen

Die Änderungen werden an den entsprechenden Quellelementen vorgenommen. Bei einer neuen Datei ist folgende Zeile an den Anfang zu schreiben:

```
/*****************************************************************
$Log$
*
*****************************************************************/
```

[45] siehe Abschnitt 3.2.1 Workbench als IDE
[46] siehe Abschnitt 5.3

Dadurch werden alle notwendigen Änderungsbeschreibungen, die über den PVCS Version Manager beim „Check-In“[47] eingepflegt werden müssen, automatisch in die Datei-Köpfe übernommen. Entwickler, Datum und Revision werden automatisch vom VM beim „Check-In“ vergeben.

Zum Beispiel:

```
/*****************************************************************
$Log:   X:\ARCHIVE\ABP\DB\SRC\ABDUEBU.PvC  $

Rev 1.1   14 Feb 1996 13:33:00   <USER>
<Änderungs-Beschreibung>
*
*
*****************************************************************/
```

e) Kompilierung über die Workbench

Die geänderten KE werden kompiliert und - falls keine Übersetzungsfehler auftreten - das Modul neu erstellt. Die entsprechenden *.EXE, *.DLL, *.LIB Dateien werden direkt in der Umgebung des Entwicklers zur Verfügung gestellt. Sämtliche Objekte und Zwischendateien befinden sich auf dem zentralen Objektbereich X:\OBJECTS*. Die Quellelemente werden vom Entwicklerbereich (U:\PRORIS) und, falls dort nicht vorhanden, vom zentralen Referenzverzeichnis X:\PRORIS gelesen.

f) Integrationstest

Die Änderungen werden getestet. Bei fehlerfreien Test geht es weiter mit Schritt „g“. Im Fehlerfall wird zu Schritt „d“ verzweigt.

g) Anlegen Meldungs-Folder

Um den Bezug zwischen einer Meldung und den geänderten KE herstellen zu können, wird ein neuer Folder angelegt[48]. Alle geänderten KE, die mit der Meldung in Zusammenhang stehen, werden dem Folder zugeordnet. Dazu werden alle für die entsprechende Meldung geänderten Dateien in dem Folder Files Locked by <USER> markiert und über „Drag and

[47] siehe auch Schritt h)
[48] siehe Abschnitt 5.2.4

Drop“ in den neu angelegten Meldungs-Folder kopiert (incl. *.MAK, *.STS, *.DEF-Dateien).

Das korrekte Anlegen des Meldungs-Folders und die Vergabe des richtigen Versionlabels ist Voraussetzung für die spätere Integration in die Kundenumgebungen. Dateien, die nicht im Meldungsfolder stehen bzw. nicht richtig gelabelt sind, werden auch nicht in eine Version integriert.

h) Check In der geänderten Sourcen

Alle für den entsprechenden Auftrag geänderten KE werden mit Hilfe des „Check-In“ Kommandos archiviert. Beim „Check-In“ muß ein Änderungstext eingetragen werden. Sollen alle Dateien den gleichen Änderungstext erhalten, so kann man die entsprechende Option auswählen, ansonsten wird für jede Datei einzeln der Änderungstext nachgefragt. Zusätzlich wird an alle geänderten Dateien der Foldername als Versionlabel vergeben. Sollten sich Dateien nicht geändert haben (z.B. *.DEF oder *.MAK-Files), so wird dies vom VM erkannt und diese Dateien werden nicht erneut archiviert. Dadurch erhalten nur die tatsächlich geänderten Dateien eine neue Revisions-Nummer.

Nach Beendigung dieser Aktion werden noch einmal alle bearbeiteten Dateien angezeigt und sind für alle anderen Entwickler wieder freigegeben (Sie erscheinen nicht mehr in dem Folder Locked by <USER>). Zusätzlich werden sie aus dem Arbeitsverzeichnis des aktuellen Entwicklers gelöscht.

i) Dokumentation im PVCS-Tracker

Damit sind die Entwicklungstätigkeiten für diesen Auftrag abgeschlossen. Die Änderungs-Dokumentation erfolgt wiederum im PVCS-Tracker[49]. Damit ist die Bearbeitung für den Entwickler abgeschlossen.

5.2.6 Der Integrationszyklus

Im Integrationszyklus werden alle Aufgaben erledigt, die mit der Integration von PRORIS-Meldungen in eine Kundenumgebung (zukünftig eine Integrationsumgebung)[50] im Zusammenhang stehen. Ausgangspunkt ist die Auslieferung eines neuen Release oder die Korrektur (Patch) eines bestehenden Release. Der Integrationszyklus findet sich in der

[49] siehe auch Abschnitt 4.2.1 Schritt 3
[50] siehe Abschnitt 3.3 Umgebungsmanagement

Prozeßgrafik Abb. 4-2 in dem Punkt „Änderung wird in Kundenumgebung integriert“ wieder.

Für jeden Kunden ist eine Umgebung eingerichtet, in der folgende Verzeichnisse vorhanden sind:

U:\PRORIS\BIN*
U:\PRORIS\LIB*
U:\PRORIS\DB\BND*

Diese Kundenumgebung entspricht der beim Kunden tatsächlich installierten Umgebung, um Versionen zu testen und Fehler nachvollziehen zu können.

Ein Objekt-Verzeichnis „X:\OBJECTS“, wie es in der Entwicklungs-Umgebung vorgesehen ist, entfällt in den Kunden-Umgebungen vollständig, da dort die Objekt-Dateien in die Original-Verzeichnisse U:\PRORIS* generiert werden. Dafür ist es notwendig, daß in den *.MAK-Files die Parameter für die Objekt-Verzeichnisse im Gegensatz zur Entwicklungsumgebung auf U:\PRORIS* gesetzt werden müssen. Die Steuerung erfolgt über Umgebungsvariablen[51].

Ausgehend davon, daß die beim Kunden installierte Version vollständig archiviert und mit einem Versionslabel versehen ist, muß für die Integration der Änderungen folgendermaßen vorgegangen werden:

a) Ausgangspunkt Meldungsliste

Voraussetzung für eine Integration ist, daß eine Meldungsliste existiert, auf der alle zu integrierenden Meldungen aufgelistet und im Entwicklungszyklus vollständig bearbeitet worden sind.

b) Check Out der geänderten Sourcen

Die zu integrierenden Sourcen werden auf die Kundenumgebung U:\PRORIS „ausgecheckt“(Check-Out).

Dabei ist die Zuordnung zwischen der entsprechenden Meldung und den dazugehörigen KE über den Versionlabel und den Meldungs-Folder sichergestellt.

[51] siehe Abschnitt 5.3 Buildmanagement

Beispiel:

Bei dem Kunden X ist die Version 2 installiert. Aufgrund einer Fehlermeldung wurden Änderungen an einem Modul vorgenommen, die gekennzeichnet als Version 3 zum Kunden ausgeliefert werden muß. Die Änderungen wurden durch einen Entwickler ausgeführt und wieder eingecheckt. Es muß sichergestellt werden, daß es keine Inkonsistenzen an den geänderten KE gibt (z.B. wenn teilweise andere Funktionen integriert sind). Der Entwickler hat einen Folder mit der Meldungsnummer XX/311299/1.1 angelegt, in der sich folgende Dateien befinden:

TMLTEMLO.C
TMLTEMPF.C
TML.MAK
TML.STS
TML.DEF

Die entsprechenden geänderten Revisionen sind ebenfalls mit dem Versionslabel XX/311299/1.1 versehen. Der Versionsverwalter muß nun prüfen, ob zwischen dem Versionslabel 2 und dem Versionslabel XX/311299/1.1 an den entsprechenden Dateien Änderungen vorgenommen worden sind. Ist dies der Fall, so muß überprüft werden, ob diese Änderungen zu Inkonsistenzen führen können bzw. ob diese Änderungen ebenfalls integriert werden können.

Zusätzlich müssen für dieses Beispiel noch folgende Dateien (Versionslabel 2 / exklusiv) „ausgecheckt“ werden:

U:\PRORIS\BIN\TML.DLL
U:\PRORIS\LIB\TML.LIB

Die zusätzlich notwendigen KE, die nicht verändert wurden, werden automatisch mit dem richtigen Versionslabel 2 zur Verfügung gestellt.

U:\PRORIS\TMP\LOGIC\SRC*.C
U:\PRORIS\TMP\RES\SRC*.RC (*.DLG)
U:\PRORIS\TMP\INC*.*

c) Build über Workbench

Die betroffenen Module werden über die Workbench neu kompiliert. In diesem Fall sollten die Dateien TMLTEMLO.OBJ und TMLTEMPF.OBJ neu erstellt werden. Bei allen anderen KE erkennt der VM, daß keine erneute Kompilierung notwendig ist.

d) Test in Kundenumgebung

Mit den unter Schritt „c“ neu erstellten DLL’s bzw. EXE’s wird in der Kundenumgebung ein Integrationstest vorgenommen.

e) Einchecken / Labeln der geänderten Dateien

Nach erfolgreichem Test, werden die unter Schritt „b“ exklusiv „ausgecheckten“ Dateien wieder eingecheckt. Dabei wird durch den PVCS VM erkannt, daß sich einige KE nicht verändert haben. In diesem Fall braucht keine neue Revision erzeugt zu werden. Danach werden für diese Dateien die Versionslabel vergeben. Dabei wird die Nummer der neuen Version (Version 3) eingetragen.

Damit steht das neue Release oder der Patch zur Vorbereitung der Auslieferung bereit.

5.3 Build-Management

Das Build-Management (BM) beschäftigt sich mit der Definition der Regeln und Abhängigkeiten der einzelnen KE, die den Übersetzungsprozeß verläßlich und reproduzierbar gestalten. Bei PRORIS wird das BM durch das Werkzeug PVCS Configuration Builder (CB) unterstützt. Der CB besteht im wesentlichen aus der MAKE-Komponente und einigen Hilfsprogrammen. Mit der MAKE-Komponente können Regeln für die Ableitung der einzelnen Typen von KE hinterlegt werden. Im Gegensatz zu anderen MAKE-Werkzeugen kann der CB auch mit den Archiven der PVCS Version Managers umgehen. Damit können die Regeln sich direkt auf z.B. Versionslabel oder auch Revisionen beziehen, ohne daß die KE zuvor aus dem Archiv herausgeholt werden müßten. Damit läßt sich der Systemgenerierungsprozeß automatisieren. Bei der Größe des PRORIS-Archivs kommt es jedoch zu erheblichen Performanceproblemen, so daß häufig doch mit den „ausgescheckten“ KE in den Kundenumgebungen gearbeitet wird.

Den Systemgenerierungsprozeß, auch für Dritte, nachvollziehbar zu gestalten, ist ein weiterer wichtiger Aspekt. Dazu gibt es im CB entsprechende Generierungsprotokolle, die Aufschluß darüber geben, aus welchen Teilen ein Modul zusammengebaut wurde. Darüber hinaus werden die Module (EXE und DLL) mit einem sogenannten „Footprint“ versehen. Dieser „Footprint“ enthält alle KE mit ihrer Revision, die in das entsprechende Modul eingeflossen sind. Der „Footprint“ befindet sich direkt in der Datei des Moduls. Somit können auch bei Kunden ausgelieferte Releasestände nachträglich auf ihre Zusammensetzung hin geprüft werden.

Den Systemgenerierungsprozeß (für OS/2) für alle Programme und Bibliotheken ist in Abbildung 5-4 dargestellt. Für die Reportdateien und die Dateien mit den SQL-Zugriffen kommen jeweils vom PRORIS-Team entwickelte Prä-Kompiler zum Einsatz. Ansonsten unterscheidet sich der Prozeß nicht von dem im Abbildung 5-4 dargestellten.

Der CB steht sowohl unter OS/2 als auch unter Windows mit gleicher Funktionalität zur Verfügung. Durch die Flexibilität des CB ist es einfach, zwei unterschiedliche Betriebssystemplattformen zu unterstützen. Die Unterschiede zwischen den Plattformen bezüglich des Systemgenerierungsprozesses, können in zentralen Konfigurationsdateien hinterlegt werden. Der Entwickler ist insofern von diesem Spezialwissen befreit. Die zentrale Konfiguration gewährleistet zusätzlich, daß jeder Entwickler mit denselben Einstellungen kompiliert und bindet. Damit wird eine gefährliche Fehlerquelle ausgeschaltet.

Der CB unterstützt nicht die Zusammenstellung bzw. die Auslieferung von Releaseständen. Daher sind vom PRORIS-Team für die Verwaltung von ausgelieferten

Releaseständen eigene Verfahren entwickelt worden. Es werden z.B. Listen darüber geführt, welcher Kunde welches Release im Einsatz hat. Darüber hinaus gibt es Werkzeuge die einen Releasewechsel bei einem Kunden automatisieren.

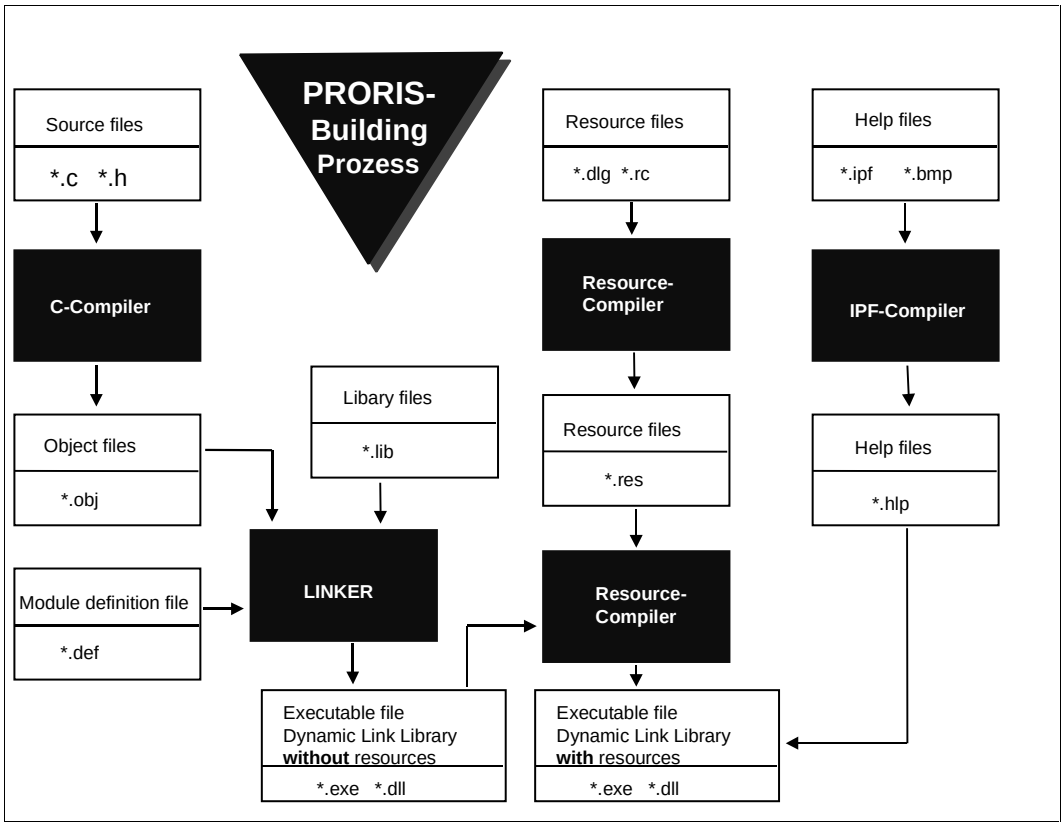


Abb. 5-4: PRORIS Buildingprozeß

5.4 Bewertung des Versionsmanagements

Das Versionmanagement in PRORIS hat viele Facetten. Auf der einen Seite gibt es die Dateiverwaltung mittels PVCS Version Manager, der ein umfangreiches und ausgereiftes Werkzeug für die Dateiverwaltung darstellt. Auf der anderen Seite finden sich Varianten im Sourcecode, die entweder durch bedingte Übersetzung oder zur Laufzeit über Konfigurationsdateien angesteuert werden. Dieses erscheint zunächst komplex und aufwendig, muß aber bei genauerer Betrachtung als sinnvoll und notwendig erachtet werden, da ansonsten die Verwaltung im PVCS-Version Managers erheblich komplexer und aufwendiger werden würde. Vor dem Hintergrund der großen Anzahl von Konfigurationselementen in PRORIS würde dies zu längeren Wartezeiten im Entwicklungszyklus führen. Der dateibasierte Ansatz des PVCS-Version Managers führt ohnehin schon zu Performanceproblemen[52].

Leider sind zur Zeit einige wichtige Konfigurationselemente nicht unter der Kontrolle der Versionsverwaltung. Dazu gehören die Kompiler der Fremdhersteller, Spezifikationen im Word-Format, Betriebssystem-Bibliotheken sowie Testdaten und Testfälle. Diese KE sollten dringend der Versionsmangement unterworfen werden, weil sie Einfluß auf die Gesamtkonfiguration haben. Dieses gilt insbesondere für die Kompiler und Bibliotheken, die sogar direkt am Systemgenerierungsprozeß beteiligt sind.

Eine Schwäche ist die unzureichende Integration mit PVCS-Tracker. Dies führt auf Seiten der PVCS Version Managers dazu, daß die PRORIS-Meldungen als Folder abgebildet werden und auch entsprechend gelabelt werden muß. Dabei wird mit organisatorischen Maßnahmen versucht die Mängel des SKM-Systems auszugleichen. Diese organistorischen Maßnahmen sind aufwendig und fehleranfällig. Sie fügen sich nicht nahtlos in den Entwicklungsprozeß ein und bilden somit eine Quelle der Unzufriedenheit bei den Projektbeteiligten.

Auf Seiten des Buildmanagements finden sich nahezu alle Anforderungen aus Abschnitt 2.5 wieder. Es wird lediglich kein differenziertes Auslieferungsverfahren unterstützt, da im Versionsmanagement nicht vollständig bzw. differenziert genug Versionslabel an die Dateien angehängt werden. Dies geschieht vor allem aus Performancegründen, da das Anhängen von Versionslabeln an alle Dateien im PRORIS-Archiv mehr als 12 Stunden in Anspruch nimmt. Dieses Performanceproblem führt wiederum zu einem erhöhten Verwaltungsaufwand auf Seiten des Versionsmanagements, da dadurch die Patches und Zwischenversionen, in den jeweiligen Kundenumgebungen abgebildet werden müssen.

[52] siehe auch Abschnitt 3.6

Dadurch ist es nicht mehr möglich jede ausgelieferte Version des Systems automatisch aus dem PRORIS-Archiv wiederherzustellen. Dies kann nur durch manuelle Eingriffe erfolgen.

6 Bewertung und Ausblick

In dieser Diplomarbeit wurde zunächst ein Einblick in das Software-Konfigurationsmanagement (SKM) gegeben. Es wurde deutlich, daß SKM-Systeme eine bedeutende Rolle im Rahmen der Softwareentwicklung spielen, und in der Zukunft noch wichtiger werden.

Beim dem SKM-System von PRORIS wurde deutlich, daß es durchaus Zielkonflikte zwischen den Vertretern des SKM und den Entwicklern oder auch dem Projektmanagement gibt.

Der Projektmanager schaut auf seine Ressourcen in Form von z.B. Kosten und Personen. Der Aufbau eines umfangreichen SKM-Systems bindet zunächst Ressourcen, die für die Entwicklung des Softwaresystems nützlich wären. Hier muß Überzeugungsarbeit geleistet werden, die deutlich macht, daß dieser Aufwand sich in den späteren Phasen des Projektes, z.B. beim Testen, positiv auswirkt. Dies wird bei einem Projekt wie PRORIS, das bereits seit Jahren bei Kunden in Produktion läuft, und somit einen hohen Wartungs- aber auch Entwicklungsanteil hat, besonders deutlich.

Die Entwickler fühlen sich durch ein SKM-System häufig in ihrer Freiheit eingeschränkt. Sie sehen zwar die Notwendigkeit solcher Systeme, fühlen sich aber durch den entstehenden Mehraufwand behindert. An dieser Stelle ist es wichtig die Entwickler bei dem Design des Entwicklungsprozesses aktiv zu beteiligen, um etwaige Hürden rechtzeitig zu erkennen und gemeinsam das SKM-System handhabbar zu gestalten. Diese Aufgabe bewegt sich immer in dem Spannungsfeld, auf der einen Seite möglichst viele detaillierte Informationen sammeln zu wollen und auf der anderen Seite den Entwicklungsprozeß nicht zu behindern. Dies kann nur mit effektiven Werkzeugen erreicht werden, die es erlauben, die unterschiedlichen Disziplinen des SKM, wie Versionsmanagement, Buildmanagement und Änderungsmanagement miteinander zu integrieren. Das derzeitige SKM-System von PRORIS versucht einige der Schwächen der SKM-Werkzeuge durch organisatorische Maßnahmen auszugleichen. Es hat sich jedoch gezeigt, daß gerade diese organisatorischen Maßnahmen zu Akzeptanzproblemen führen, da sie den SKM-Prozeß aufblähen. Eine weitere Schwierigkeit bei dem SKM-System von PRORIS ist die Entwicklung für eine unzureichend unterstützte Betriebssystemplattform (OS/2). Diese inzwischen weniger verbreitete Plattform bewirkt den Einsatz veralteter SKM-Werkzeuge, die noch nicht über die geforderten Integrationsmöglichkeiten verfügen. Zusätzlich sorgt diese Plattform für eine hohe Komplexität in der Entwicklungsumgebung. In Zukunft wird diese Plattform nicht mehr unterstützt werden. Dies wird Anlaß für die Erneuerung des SKM-Systems von PRORIS sein.

Der Markt für SKM-Werkzeuge ist bisher durch viele Einzellösungen für die jeweiligen Disziplinen im Entwicklungszyklus bestimmt. Diese Lösungen arbeiten häufig nur ungenügend zusammen. Es setzt sich aber zunehmend ein integrativer Ansatz durch, bei dem mit Hilfe der Werkzeuge des SKM folgende Abläufe unterstützt werden sollen:

- Pflichten- Lastenheft für das zu erstellende Programm,
- Verwaltung der Metadaten zur Programmierung,
- Programmentwicklung,
- Test: Funktionstest, Lasttest, Verfügbarkeitstest, Performancetest,
- Softwareverteilung mit Rückschlüssen auf den Benutzerbedarf, der Hardwarekonfiguration etc.,
- Helpdesk und Support durch Bereitstellung der Daten über Hardware- und Software-Konfiguration und
- Rückführung der am Helpdesk gewonnen Informationen in den folgenden Software-Update-Zyklus.

Die heutigen Werkzeuge decken bei weitem nicht diese Anforderungen ab. Es muß aber auch nicht der gesamte Leistungsumfang in einem Werkzeug abgedeckt werden. Jedoch leistungsfähige Schnittstellen zwischen den einzelnen Werkzeugen müssen vorhanden sein.

Die Beobachtung des Marktes und auch diese Arbeit zeigen, daß SKM-Systeme nicht mehr nur aus Versionsmanagement-Werkzeugen bestehen können, sondern mächtige prozeßorientierte Verwaltungswerkzeuge werden, die den gesamten Softwareentwicklungszyklus unterstützen sollen.

Literatur

[Bers 80] Bersoff E., Henderson V., Siegel S.
Software Configuration Management
Prentice-Hall 1980

[Boehm 88] Boehm B.
A spiral model of software development and enhancement
Computer Magazine, Mai 1988

[Coll 87]. Collofello J., Buck J.
Software Quality Assurance for Maintenance
IEEE Software, September 1987

[Compton 94] Compton S., Conner G. R.
Configuration Management for Software
Van Nostrand Reinhold 1994

[DGQ 12-51] DGQ-NTG-Schrift 12-51,
Software-Qualitätssicherung; Aufgaben, Möglichkeiten, Lösungen
Beuth, Berlin 1986

[Dunn 93] Dunn, Robert H.
Software-Qualität, Konzepte und Pläne
Hanser, 1993

[Geratewohl 75]. Geratewohl,
Rückversicherung
1975

[Grossmann 77]. Grossmann, Marcel
Rückversicherung eine Einführung
Verlag Peter Lang, Bern, Frankfurt am Main 1977

[Höft 85] Höft D., Schaller H.
Software-Konfigurationsmanagement in großen Softwareprojekten
Informatik-Spektrum 8, pp. 576-584

[IEEE 88] IEEE
Guide to Software Configuration Management
1988

[ISO 10007] Deutsches Institut für Normung e.V.
Qualitätsmanagement: Leitfaden für Konfigurationsmanagement
Deusche Fassung prEN ISO 10007:1995
Berlin; Köln: Beuth

[IT Fokus 7/2000] Dave Robertson
Auswahl und Einsatz von Softwarekonfigurations-Tools
IT Fokus Ausgabe Juli 2000
IT Verlag 2000, Höhenkirchen

[Oestereich 99] Bernd Oestereich
Erfolgreich mit Objektorientierung
Oldenbourg Wissenschaftsverlag, 1999

[Ovum 99] Clive Burrows, Ian Wesley
Ovum Evaluates Configuration Management
IT Research, 1999

[Pfeiffer 75]. Pfeiffer, Christoph
Einführung in die Rückversicherung
Gabler 1977

[PRORIS 95]. PRORIS Broschüre
1995

[PRORIS 99] PRORIS Systemdokumentation
1999

[Scheller 96] Scheller B.
Methoden des Qualitätsmanagement als Instrument für effiziente Softwareentwicklung, Studienarbeit 1996

[Tichy 88] Tichy W.
Tools for Software Configuration Management
Softwaretechnik-Trends, Heft 8-1, 1988, pp.51-70

[Wallmüller 90] Wallmüller, Ernest
Softwarequalitätssicherung in der Praxis
Hanser, München 1990

[Whitgift 91] Whitgift D.
Methods and tools for software configuration management
John Wiley & Sons, West Sussex 1991

Anhang A: Anforderungsmatrix

Ergebnis einer Befragung des PRORIS-Projektteams

Legende:
3 = notwendig
2 = wichtig
1 = wünschenswert
0 = irrelevant

Die Werte in "(...)" dokumentieren die von der Befragung abweichende Meinung des Autors

		Kunden sicht	**Entwickler -Sicht**	**PM-Sicht**	**QM-Sicht**
1. Übergreifende Anforderungen					
1.1 Allgemeine Anforderungen					
1.1.1 Benutzerfreundlichkeit		2	2	2	2
1.1.2 Performance / Skalierbarkeit		2	2	2	2
1.1.3 Zugriffsschutz / Rollenkonzept		2	3	1	2
1.1.4 Datensicherung		2 (3)	3	3	3
1.1.5 Plattformunabhängigkeit		0	2	1	2
1.1.6 Anpaßbarkeit / Erweiterbarkeit		0	2	1	2
1.1.7 Flexible konfigurierbare Reports		2	1	2	2
1.2 Teamunterstützung					
1.2.1 Verteilte Softwareentwicklung		0	3	2	2
1.2.2 Verteilte Datenhaltung		0	2	1	1
1.2.3 Unterstützung von Web-Technologie		2	2	2	1
1.3 Prozeßunterstützung					
1.3.1 Flexible Modellierung der Entwicklungsprozesse		0	2	2	2
1.3.2 Steuerung und Kontrolle der Entwicklungsprozesse		0	2	2	2
1.3.3 Automatisierung von Prozessen		0	2	2 (3)	2 (3)
1.4 Integration					
1.4.1 Integration mit Entwicklungswerkzeugen		0	3	1	2

		Kunden sicht	Entwickler -Sicht	PM-Sicht	QM-Sicht
1.4.2 Integration mit Testwerkzeugen		0	2	1	2
1.4.3 Integration mit Projektmanagementwerkzeugen		0	1	2	1
2. Funktionsspezifische Anforderungen					
2.1 Versionsmanagement					
2.1.1 Sicheres Ein- und Auschecken der KE		0	3	1	3
2.1.2 Unterstützung von Promotionsstufen		0	1	2	2
2.1.3 Prozeßunterstützung (Event Trigger)		0	2	2	2
2.1.4 Umgebungsmanagement		0	2	1	2
2.1.5 Unterstützung mehren Entwicklungslinien (merge)		0	3	2	2
2.1.6 Verwaltung unterschiedlichster KE		0	1	2	2
2.1.7 Komfortable Abbildung der Produktstruktur		0	2	2	2
2.1.8 Integration mit dem Änderungsmanagement		0	2	2	3
2.2 Änderungsmanagement					
2.2.1 Flexibel gestaltbare Änderungs-/Fehlermeldungen		2	1	2	2
2.2.2 Integration mit der Versionsmanagement		0	2	2	3
2.2.3 Prozeßunterstützung		0	1	2	2
2.2.4 Trend- und andere Analysereports		0	1	2	2
2.2.5 Releasenotes		2	2	2	2
2.2.6 Statusreports		1	2	2	2
2.2.7 Einbindung von beliebigen Dokumenttypen		2	2	2	2
2.3 Buildmanagement					
2.3.1 Regelbasierte Abhängigkeiten der KE		0	3	1	3
2.3.2 Integration mit dem Versionsmanagement		0	2	1	3
2.3.3 Unterstützung des Auslieferungsverfahrens (full/delta)		1 (2)	2	2	3
2.3.4 Unterstützung von Footprinting (Stücklisten)		0	2	1	3

Anhang B: PRORIS-Meldung

PRORIS - Meldung

An Consult TEAM Hamburg GmbH FAX-Nr. 0 40 / XXX
TEL-Nr. 0 40 / YYY

Legende

Typ		Priorität			Wo gefunden	
Fehlermeldung	F	sofort	showstopper	1	Testumgebung	T
Verbesserung	V	gleicher Arbeitstag	hoch	2	Produktionsumgebung	P
Erweiterung	E	(wenn Meld. bis 13.00 Uhr)			irrelevant	K
Prüfung	P	5 Arbeitstage	mittel	3		
		10 Arbeitstage	niedrig	4		
		irrelevant	keine	5		

Kunde: _____ **Datum: ___/___/___** **Lfd.-Nr./Nachtragsnr.: _____/_____**

Typ: _____ Priorität: _____ Wo gefunden: _____

Ansprechpartner: _____ PRORIS-Version: _____

Anwendung:

Titel: _____

Gewünschte Fertigstellung: ___/___/___ Abgenommen am: ___/___/___

Beschreibung:

Anhang C: Legende zur PRORIS-Meldung

Kunde:	ww	Kunde ww
	xx	Kunde xx
	yy	Mitarbeiter yy (interne Meldungen)
	zz	Mitarbeiter zz (interne Meldungen)
Meldungstyp:	F	Fehler
	V	Verbesserung
	E	Erweiterung
	P	Prüfung
	FM	Falschmeldung
	A	Auftrag
Priorität:	showstopper	Sofortige Rückmeldung => **nur bei Meldungstyp Fehler**
	hoch	Rückmeldung am gleichen Arbeitstag wenn die Meldung bis 13.00 Uhr eingegangen ist. Bei Eingang nach 13.00 Uhr wird am nächsten Arbeitstag reagiert.
	mittel	Rückmeldung innerhalb von 5 Arbeitstagen
	niedrig	Rückmeldung innerhalb von 10 Arbeitstagen
	keine	irrelevant
Wo gefunden:	T	Testumgebung
	P	Produktionsumgebung
	K	irrelevant
	C	Code-Inspektion (nur von CTH)
	S	System-Test (nur von CTH)
Anwendung:	xxp	Anwendung xx
	yyp	Anwendung yy