Konzeption eines Software-Lifecycle-Managementsystems (SLM) zur Unterstützung und Beschleunigung von Softwareentwicklungsprozessen

Dem Fachbereich 12 Maschinenwesen der
Universität GH Essen zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
vorgelegte Dissertation

von
Jens Stolpmann
aus Essen

Tag der mündlichen Prüfung: 29.10.2003

Gutachter:

1. Prof. Dr.-Ing. H. J. Stracke
2. Prof. Dr.-Ing. D. Bergers

Vorwort

Diese Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Ingenieurinformatik im Fachbereich Maschinenwesen der Universität Gesamthochschule Essen.

Herrn Univ.-Prof. Dr.-Ing. H. J. Stracke danke ich für die Anregung und Förderung dieser Arbeit und den mir stets gewährten Freiraum bei der Durchführung, sowie die Übernahme des Erstgutachtens.
Herrn Univ.-Prof. Dr.-Ing. D. Bergers danke ich für das meiner Arbeit entgegengebrachte Interesse und für die Übernahme des Korreferats.

Herrn Dr.-Ing. Frank Lobeck danke ich für sein stets offenes Ohr beim Schreiben der Arbeit.

Den derzeitigen und früheren Kollegen sowie den Studierenden am Institut für Ingenieurinformatik danke ich für die gute Zusammenarbeit und die Anregungen, die aus unserer gemeinsamen Arbeit hervorgegangen sind.

Ebenso danke ich meinen Eltern und meiner Schwester Karin für ihre Unterstützung und Hilfe.

Essen, im August 2003 Jens Stolpmann

Inhaltsverzeichnis

1 Einleitung 10
1.1 Software als Produkt 11
1.2 Aufgabenstellung und Realisierung 14
2 Softwareentwicklung heute 16
2.1 Allgemeines 16
2.2 Konkrete Fallbeispiele aus der Praxis 18
2.2.1 Softwareentwicklung im CAD-Bereich 20
2.2.2 Softwareentwicklung im PDM-Umfeld 21
2.2.3 Softwareentwicklung im SAP-Umfeld 23
2.2.4 Softwareentwicklung im PLM-Umfeld 27
2.3 Beurteilung der Defizite der analysierten Praxisbeispiele 29
2.3.1 Projektabwicklung 29
2.3.2 Eingesetzte Tools 29
2.3.3 Qualitätssicherung 32
2.3.4 Dokumentation und Kundeninformationen 32
2.3.5 Wartung und Weiterentwicklung 33
2.4 Zusammenfassung der momentanen Defizite in der Softwareentwicklung 34
3 Forderungen an ein Software-Lifecycle-Managementsystem 39
3.1 Umfang des SLM 40
3.2 Anforderungen an das Datenmanagement 41
3.2.1 Verwaltung des Quellcodes 41
3.2.2 Realisierung eines Konfigurationsmanagements 44
3.2.3 Support- und Dokumentationsmöglichkeiten erstellter Software 44
3.3 Anforderungen an das Prozessmanagement 48
3.4 Unterstützung der Kommunikation von verschiedenen Entwicklern 51
3.5 Oberfläche des SLM 53
3.6 Berücksichtigung unterschiedlicher Programmiersprachen 54
3.7 Zusammenfassung der Anforderungen 55
4 Aktuelle Technologien zur Verbesserung des Softwareentwicklungsprozesses 57
4.1 Allgemeines 57

4.2 Abgrenzung zu PLM-Systemen 57
4.3 Grenzen von Datenbanksystemen 60
4.4 Möglichkeiten von CASE-Tools 62
4.5 Neue Technologien der Software-Entwicklung 64
4.6 Ausgewählte Technologien und Werkzeuge für das Konzept 71
5 Konzept für das SLM-System 75
5.1 Aufbau des SLM-Systems 75
5.1.1 Client Server Architektur 76
5.1.2 Trennung von Datenhaltung und Prozesssteuerung 77
5.1.3 Systemkommunikation / Datenaustausch 78
5.2 Datenbankmanagementsystem 79
5.2.1 Projektstruktur 80
5.2.2 Basis des Datenmodells 83
5.2.3 Funktionsweise der Links 85
5.2.4 Versionisierung 86
5.2.5 ER-Diagramm der Datenbank 88
5.3 Server 89
5.3.1 Zugriff auf die Datenbank 91
5.3.2 Aufbau der Klassen 95
5.3.3 Lebenszyklus 96
5.3.4 Anlage von Metadokumenten 100
5.3.5 Workflow 101
5.3.6 Installationsversionen 105
5.3.7 Benutzerverwaltung und Zugriffssteuerung 108
5.3.8 Bereitstellung von Dateien 111
5.3.9 Scriptsprache 111
5.3.10 Klassifizierung 112
5.3.11 Kommunikation mit dem Client 113
5.4 Client 115
5.4.1 Benutzerführung 115
5.4.2 Anmeldung/Abmeldung am SLM-System 118
5.4.3 Projekte auf den Client laden 118

5.4.4 Dateien Ein-/Auschecken 119
5.4.5 Dateiüberwachung 121
5.4.6 Integration 121
5.4.7 Schnittstellen zu CASE-Tools 123
5.4.8 Schnittstellen zu Office-Software 124
5.4.9 Schnittstellen zu Code-Ausführungssoftware 124
6 Beispielhafte Anwendung des SLM-Konzeptes 126
6.1 Auswahl eines Anwendungsbeispiels 126
6.2 IST-Zustand der Softwareentwicklung PP2000 128
6.3 Struktur und Aufbau von PPNEU 140
6.3.1 Dateiaufbau 146
6.3.2 Inhalte der Metadokumente 147
6.4 Programmsteuerung von PPNEU 149
6.4.1 Startvorgang 149
6.4.2 Ignorieren fehlerhafter Makros 150
6.4.3 Konfigurationsmanagement 152
6.4.4 Möglichkeiten der Lizensierung 153
6.4.5 Verbesserungen der Support-Möglichkeiten 154
6.5 Integration der Entwicklungsumgebung für PPNEU 157
6.5.1 Schnittstelle zum Editor 159
6.5.2 Schnittstelle zu ME10 161
6.5.3 Ein- und Auschecken einzelner Makros 162
6.6 Vorgehensweise bei der Durchführung des Projektes PPNEU 164
6.6.1 Definition des Projektteams 164
6.6.2 Freigabeschema 166
6.6.3 Prozessdefinition für das Projekt PPNEU 167
6.6.4 Implementierung der Benutzerebene 168
6.6.5 Implementierung der Funktionsebene 172
6.6.6 Integration des Systems 173
6.6.7 Test und Fehlerbehebung 174
6.6.8 Auslieferung an den Kunden 176
7 Zusammenfassung und Ausblick 178

8 Anhang 180
8.1 Literaturverzeichnis 180
8.2 Abbildungsverzeichnis 184

1 Einleitung

Im Zuge der weltweiten Globalisierung wird der Druck auf die produzierenden Unternehmen im Hinblick auf Zeit, Kosten und Qualität ständig größer. Dies hat zur Folge, dass die Produktionen von einer immer strengeren Projektabwicklung begleitet werden müssen. Jede Art von nicht vorhergesehener Unterbrechung in der Produktionsabfolge führt zwangsläufig zu einer Nichteinhaltung des Zeit- und Kostenrahmens, der in der Regel, bedingt durch den enormen Konkurrenzdruck, nur äußerst unzureichend in den vertraglichen Vereinbarungen zwischen Lieferant und Kunde enthalten ist. Die damit verbundene Nichteinhaltung von Terminen und Kosten kann bei beiden Vertragspartnern zu Kapazitätsengpässen führen oder beim Produzenten Liquiditätsprobleme auf Grund von eventuell anfallenden Konventionalstrafen nach sich ziehen.

Abhilfe kann hier nur durch ein streng organisiertes Projektmanagement erreicht werden. Hierbei sind unbedingt alle Bereiche der Wertschöpfungskette eines Produktes von der Planung über die Herstellung bis zu der Vernichtung mit einzubeziehen. Diese schwierige Aufgabe wird in den letzten Jahren zunehmend durch die EDV unterstützt. Hierbei werden für die Bereiche Planung, Kontrolle und Steuerung unter anderem die Software-Produkte MS-PROJECT eingesetzt.

Der Einsatz derartiger Software-Produkte ist heute bereits ein fester Bestandteil in allen Bereichen eines Unternehmens. Jeder Mitarbeiter hat sowohl positive als auch negative Erfahrungen im Umgang mit den elektronischen Medien erlebt. Positiv, wenn z.B. in kürzester Zeit ein äußerst komplexes Festigkeitsproblem mit Hilfe der Finite Element Methode gelöst wird und negativ, wenn mit einem nicht erklärbarem Absturz der Anwendungssoftware der Verlust der zeitaufwändig erzielten Daten einhergeht. Die Liste von positiven und negativen Erfahrungen in der Anwendung von Software ließe sich beliebig erweitern.

Diese Tatsache verdeutlicht die ganze Komplexität der Softwareentwicklung. In diesem Zusammenhang stellen sich eine Vielzahl von Fragen. Nachfolgend nur eine kleine Auswahl:

- Wie kann eine Software überhaupt entwickelt werden?
- Wie zuverlässig und anwendergerecht ist die Lösung?
- Wie hoch sind der Entwicklungsaufwand und die verursachten Kosten?
- Wie lässt sich eine Software durch eine andere ersetzen?
- Wie kann eine Software bewertet werden?

Der Auszug der möglichen Fragestellungen zeigt die Ähnlichkeit der Fragestellungen z. B. bei der Beurteilung von herkömmlichen Produkten aus der Fertigungsindustrie. Die Softwareentwicklung ist ein umfassender Prozess, bei welchem eine Vielzahl von Algorithmen im Allgemeinen mit mehreren Personen für andere Benutzer auf der Basis organisatorischer Richtlinien zu lösen sind. Hierbei handelt es sich um eine kreative Aufgabe, die eine äußerst effiziente Lösung zum Ziel hat, die möglichst automatisiert abläuft. Dies wiederum lässt den Schluss zu, dass die Entwicklung von Software-Produkten nicht nur eine einfache Projektbearbeitung ist, sondern den Erfordernissen eines umfangreichen Projektmanagements unterliegt.

1.1 Software als Produkt

Als Produkt wird im allgemeinen etwas Materielles angesehen, was ein Interessent in irgendeiner Form irgendwo erwerben kann. Diesem einschränkenden Verständnis für den Begriff des Produktes widerspricht die VDI-Richlinie 2221 [1] eindeutig durch folgende umfassende Definition:

> *Erzeugnis, das als Ergebnis des Entwickelns und Konstruierens hergestellt oder angewendet wird. Das können materielle (z. B. Maschinen, Verfahren) oder auch immaterielle Erzeugnisse (z. B. Programme) sein.*

Dem gegenüber gibt es eine ganze Reihe von Definitionen für den Begriff Software-Produkt, die sich inhaltlich von der VDI-Richtlinie nur unwesentlich unterscheiden, wie beispielhaft die Definition von Reiner Dumke [2] zeigt:

Ein Software-Produkt ist die Gesamtheit von Softwarekomponenten (Programmen, Dokumentationen, usw.), die als Ganzes entwickelt, vertrieben, angewendet und gewartet werden.

Für herkömmliche, materielle Produkte wird für die Marktpräsenz des Produktes der Begriff Lebenszyklus verwendet, der sich in die 4 Phasen: Einführungsphase, Wachstumsphase, Reifephase und Altersphase gliedert. Im Gegensatz dazu wird der Lebenszyklus von Software-Produkten wie folgt verwendet [2] Software-Produkt:

Als Softwarelebenszyklus wird die gesamte Lebensdauer eines Software-Produktes von seiner Entwicklung über seinen Betrieb bis hin zu seiner „Außer-Betriebnahme“ bezeichnet.

Das heißt, der Lebenszyklus umfasst in einem ganzheitlichen Ansatz nicht nur die Entwicklung und Nutzung des Produktes sondern auch dessen Weiterentwicklung und Beseitigung bzw. Erneuerung. Gleichgültig ob es sich um ein materielles oder ein immaterielles Produkt handelt, vor dem Erwerb eines Produktes unterliegt das jeweilige Produkt einem so genannten Produktentwicklungsprozess.

Abbildung 1.1: Produktentwicklungsprozess [3]

Dieser Prozess sieht gemäß Abb. 1.1 für ein herkömmliches, physisches Produkt im wesentlichen folgendermaßen aus: Ausgehend von der Planung über die Entwicklung, die Fertigung und den Versand wird das so entstandenen Produkt der Anwendung überführt und solange genutzt bis es z.B. aus Rentabilitätsgründen oder wegen Unbrauchbarkeit vernichtet wird. Die während der Verwendung einsetzende Wartung sorgt nur dafür, dass das Produkt in seiner Substanz erhalten bleibt und die von ihm geforderten Funktionalitäten jederzeit erfüllt werden..

Im Vergleich dazu sieht der Entwicklungsprozess (Abb. 1.2) für ein Software-Produkt folgendermaßen aus:

Abbildung 1.2: Softwareentwicklungsprozess (vereinfacht) [4]

Auf den ersten Blick unterscheiden sich diese beiden Prozesse lediglich dadurch, dass der Softwareentwicklungsprozess keine „richtige“ Fertigung enthält [5]. Die „Fertigung“ in diesem Prozess lässt sich nicht eindeutig von der Entwicklung trennen, da die Arbeit in diesen beiden Teilbereichen iterativ abläuft. Streng genommen beschränkt sich hier die Fertigung auf die Vervielfältigung des Ergebnisses. Der wesentliche Unterschied zwischen den beiden Prozessen wird durch die beiden letzten Teilbereiche Anwendung und Wartung ausgelöst. Der Softwareentwicklungsprozess ist streng genommen durch die Auslieferung noch nicht beendet, da durch die Anwendung erhebliche Änderungen und Erweiterungen von dem Produkt gefordert werden können, was wiederum nicht nur den Erhalt der Funktionalitäten bedeuten kann sondern auch zu einer völligen Neuentwicklung führen kann.

Dies bedeutet, dass im Gegensatz zu dem theoretischen Modell des Entwicklungsprozesses mit seinen einzelnen Phasen, wie oben beschrieben, der tatsächliche Software-entwicklungsprozess in der Regel keine lineare Kette, sondern ein stark iterativer Prozess ist, bei dem es im ungünstigsten Fall dazu kommen kann, dass von der Anwendungsphase wieder in die Anforderungsphase gesprungen werden muss, wenn sich z.B. in der Anwendungsphase herausstellt, dass die der Entwicklung zugrunde liegenden Anforderungen nicht der Realität entsprechen. Gerade für dieses Problem gilt es Lösungen zu finden, die den Entwicklungsprozess unterstützen helfen.

1.2 Aufgabenstellung und Realisierung

Wie bereits ausgeführt, unterliegen die meisten Unternehmen einem hohen Konkurrenz- und somit auch einem hohen Kostendruck. Dies zwingt sie dazu, ihre Produktionsabläufe zu rationalisieren. Hierzu wurden häufig im Laufe der Jahre in der Produktion, der Verwaltung und der Entwicklung verschiedene Werkzeuge der Informationstechnologie eingesetzt, um die Arbeitsabläufe der Mitarbeiter im Unternehmen effizienter zu gestalten. Zu diesen Werkzeugen zählen z.B. CAx-Systeme für die Entwicklung, ERP[1]-Systeme für die Produktionsplanung und –steuerung, wie beispielsweise SAP-R3 [6]. Für viele dieser Software-Produkte wurden kundenspezifische Anpassungen realisiert, die mehr oder weniger ad hoc aus der Situation heraus von eigenen Mitarbeitern oder externen Dienstleistern durchgeführt wurden. Der Schwerpunkt der ausgeführten Tätigkeiten konzentrierte sich immer auf den Bereich der Software-Entwicklung selbst und weniger auf die kostentreibenden Bereiche, die danach kommen.

Der damit verbundene erhöhte Kostendruck erfordert eine Integration der gesamten Informationsverarbeitung innerhalb der Unternehmen, um dadurch möglichst optimale Produktions- und Geschäftsprozesse zu erreichen und jederzeit eine Kontrolle über die Abläufe im Unternehmen zu bekommen. Die im Unternehmen bereits vorhandenen Softwaresysteme besitzen häufig eine hohe Akzeptanz bei den Mitarbeitern und eine Ablösung durch neue integrative Systeme würde allein durch die notwendigen Schulungen der Mitarbeiter in Verbindung mit den langen Einarbeitungszeiten häufig auf heftige Gegenwehr stoßen. Auch die speziellen Anpassungen dieser Systeme an spezifische Anforderungen, wie etwa besondere Berechnungsroutinen oder andere Hilfsprogramme, die in die vorhandenen Systeme integriert wurden, müssen bei einer Ablösung entweder auf einer anderen Plattform neu geschrieben werden oder gehen im schlechtesten Fall ganz verloren. Ein weiteres Problem stellt die verlustfreie Übernahme der bereits vorhandenen Daten in die neuen Systeme dar, denn die Daten bilden letztendlich eine wesentliche Grundlage für den Geschäftserfolg des Unternehmens.

[1] ERP: Enterprise Resource Planning

Es gibt mittlerweile eine Vielzahl von Software-Entwicklungs-Tools auf dem Markt, deren Funktionsumfang aber nicht den ganzheitlichen Ansatz des Lebenszyklus-Konzeptes beinhalten, d.h., die Aufgaben der Tools enden mit der Fertigstellung der Software. Die Gesichtspunkte der Dokumentation und der Weiterentwicklung spielen in der Regel keine Rolle, obwohl gerade diese beiden Aspekte erheblich Kosten verursachen können. Aus diesem Grunde soll hier ein Informations- und Datenmanagementsystem zur Unterstützung und Beschleunigung von Softwareentwicklungsprozessen erarbeitet werden, welches als Pendant zu den klassischen PLM[2]-Systemen in herkömmlichen Produktentwicklungsprozessen anzusehen ist.

Zunächst sind die heutigen Softwareentwicklungsprozesse und die auf dem Markt befindlichen Tools zu analysieren und deren Eignung für einen ganzheitlichen Ansatz zu diskutieren. Hierbei werden verschiedene Softwareentwickler in der Industrie kontaktiert und deren Vorgehensweise analysiert. Das Ergebnis führt dann zwangsläufig zu einer Konzeption für einen überarbeiteten Entwicklungsprozess und damit zu einem verbesserten Ablauf in der Softwareentwicklung. Die Definition einer allgemeinen Softwarearchitektur soll die Grundlage für ein vielschichtiges Softwaretool darstellen, welches den neuen Entwicklungsprozess in vollem Umfang des Lebenszyklus unterstützt. Auf der Basis der neu generierten Softwarearchitektur werden dann exemplarisch zwei unterschiedliche Implementierungen vorgenommen, welche die Machbarkeit und Allgemeingültigkeit des Konzeptes nachweisen. Ein kurzes Fazit rundet die Arbeit ab.

[2] PLM: Product Lifecycle Management

2 Softwareentwicklung heute

2.1 Allgemeines

Für eine gezielte Analyse der Softwareentwicklung ist zunächst zu klären, was zu analysieren ist. Die Softwareentwicklung beinhaltet einerseits die erzeugten **„Produkte**“, d.h., Daten, Dokumente und Programme und andererseits die **Prozesse,** die diese sogenannten „Produkte“ erzeugen. Zu den Bestandteilen eines Software-Produktes gehören unter anderem:

- eine **Entwicklerdokumentation**, welche alle im Verlauf der Softwareentwicklung entstandenen Dokumente bezüglich der Anforderungen, Spezifikationen und programmtechnischen Umsetzung enthält,
- ein **Quellcode** der Software, welcher die vom Auftraggeber geforderten Funktionen umsetzt,
- ein **Benutzerhandbuch,** welches dem Anwender eine einwandfreie Bedienung der Software ermöglichen soll,
- ein **Administratorhandbuch**, das einem ausgewählten Personenkreis die Konfiguration und Installation der Software ermöglicht,
- ein **Installationsprogramm**, welches den Zugriff auf eventuell weitere Programme vorbereitet,
- eine **Dokumentation** für die Entwickler, welches die Beschreibung der Algorithmen, der erstellten Klassen und Funktionen sowie die Änderungshistorie enthält und eventuell
- eine **Demoversion** der Software, welche den Vertrieb bei der Vermarktung unterstützt und potentielle Anwender über den Funktionsumfang der Software informiert.

Diese „Produkte“ bzw. Dokumente werden in einzelnen aufeinander folgenden **Prozessen** gemäß Abb. 1.2 erzeugt. Diese Prozessabfolge des Softwarelebenszyklus kann idealisiert durch die folgende Grafik (Abb. 2.1) beschrieben werden:

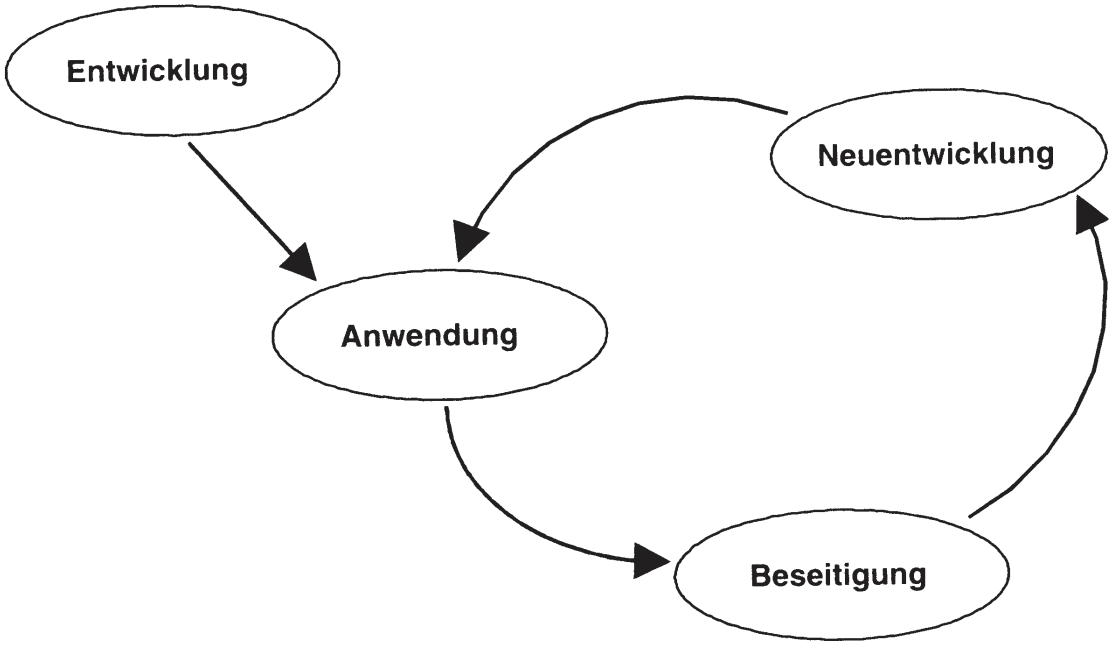


Abbildung 2.1 : Idealisierter Softwarelebenszyklus [2]

Die hier dargestellte Idealisierung entspricht allerdings nicht ganz den heutigen Gegebenheiten. Auf Grund der Komplexitäten der heutigen Aufgabenstellungen ist die Software-Entwicklung nicht mehr eine lineare Abfolge von einzelnen Projektphasen sondern durch einen sehr starken iterativen Prozess geprägt. Ebenso ist mit der Überführung von der Entwicklung in die Anwendung der Prozess nicht abgeschlossen sondern vielmehr werden in der Regel erst durch die praktische Anwendung der Software eine ganze Reihe von Weiterentwicklungen gefordert. Und nur wenn die Weiterentwicklung an Grenzen stößt, so dass sich eine Softwareverbesserung nicht mehr rechnet, führt dies zu einer Beseitigung und einer eventuellen Neuentwicklung der Software.

Für die Erstellung eines Konzeptes bezüglich einer optimalen Software-Entwicklung auf der Basis eines ganzheitlichen Ansatzes wird zunächst die aktuelle Vorgehensweise der Projektabwicklung für die Erzeugung von Software-Produkten ausführlich analysiert und gleichzeitig die erzeugten „Produkte“ einer Beurteilung unterzogen, um festzustellen, an welcher Stelle die Prozesse verbessert werden können und ob die Form der momentan erzeugten „Produkte“ den Anforderungen an ein neues Konzept genügen. Für eine Beurteilung des Ist-Zustandes bezüglich der Vorgehensweise bei der heutigen

Softwareentwicklung ist es zunächst sinnvoll einige Softwareentwickler zu kontaktieren und deren Vorgehensweise im praktischen Alltag zu analysieren.

2.2 Konkrete Fallbeispiele aus der Praxis

Bei der Auswahl der Entwickler ist nicht so sehr die Anzahl der befragten Unternehmen von Bedeutung sondern eher die Unterschiede in dem jeweils zu erstellenden Softwarespektrum, damit eine hinreichend allgemeingültige und gleichzeitig möglichst umfassende Aussage über die momentane Arbeitsweise in der Softwareentwicklung getroffen werden kann. Ausgewählt wurden vier mittelständische Unternehmen, die mit eigenen Mitarbeitern in den Abteilungen Entwicklung, Vertrieb und Support gemäß Abb. 1.2 tätig sind und das folgende Spektrum bearbeiten:

1. Kundenspezifische Softwareanpassungen im CAD-Umfeld für das Software-Produkt Solid Edge von Unigraphics.
2. Eigen- und Weiterentwicklungen einer PLM-Software und Entwicklung von Schnittstellen zu verschiedenen CAD-Systemen.
3. Konzernweite Dienstleistungen im Umfeld des ERP-Systems SAP.
4. Durchführung von Customizing-Projekten für verschiedene Branchen im PLM-Umfeld des Workmanagers von CoCreate und im CAD-Umfeld von Unigraphics.

Obwohl die einzelnen Softwareprojekte sehr unterschiedlich sind, so kann zumindest für die gemeinsame Schnittmenge ein einheitlicher Fragenkomplex vorab formuliert werden, der auf einen ganzheitlichen Softwarelebenszyklus abzielt. Während der Interviews können dann auf dieser Basis jeweils gezielt weitere ins Detail gehende Fragen gestellt werden. Hier nun ein Auszug aus dem Fragenkomplex:

1. Welche Art von Software wird entwickelt?
   - Standalone oder im Zusammenhang mit einem speziellen Produkt?
   - Wenn Mischform, wie sind die Anteile?
   - Gibt es kundenspezifische Anpassungen?

- Ist die Software eigenständig oder ein Zusatzmodul zu einer anderen Software?

2. Wie läuft die Softwareentwicklung ab?
   - Wie ist die Kundenanfrage und wer entscheidet?
   - Wer vollzieht welche Schritte?

     Spezifikation, Entwurf, Implementierung, Integration, Wartung, etc.
   - Welches Vorgehensmodell wird eingesetzt?

     Wasserfallmodell, Spiralmodell, etc.

3. Welche Tools werden eingesetzt, um die Entscheidungsprozesse zu unterstützen?

4. Wie erfolgt das Projektmanagement?
   - Wie werden die Entwickler koordiniert?
   - Welche CASE Tools werden eingesetzt?

5. Wie wird das Versions- bzw. Revisionsmanagement realisiert?
   - Kann jeder Entwicklungsstand der Software wiederhergestellt werden?

6. Wie erfolgt die Qualitätssicherung?
   - Wie wird getestet und wer testet?
   - Wird die Qualitätssicherung bereits in der Planungsphase mit einbezogen?

7. Wie ist der Kunde in den Softwareauftrag eingebunden?
   - Schon während der Projektarbeit?
   - Wie erfolgt die Abnahme durch den Kunden?
   - Wird zuerst auf einem Testrechner installiert?
   - Wie erfolgt die Produktivschaltung?

8. Wie erfolgt die Dokumentation der Software?
   - Wann wird die Dokumentation erstellt?
   - Wer erstellt sie?
   - Welche Art Dokumentation wird erstellt?

- Unterliegt auch die Dokumentation einem Versionsmanagement?

9. Wie wird die Wartung bzw. der Support abgewickelt?
   - Welche Informationen hat der Support?
   - Wie sind diese Informationen abrufbar?
   - Werden durch den Support Weiterentwicklungen veranlasst?

10. Welche Informationen erhält der Kunde über die Software.
    - Nur Anwenderinformationen?
    - Alle Informationen aus Spezifikations-, Entwurfs- und Entwicklungsphase?

2.2.1 Softwareentwicklung im CAD-Bereich

Dienstleistungsspektrum

Auf Anforderung des Marktes werden vom Dienstleister mit ca. 55 Mitarbeitern individuelle Anwendungsprogramme für CAD- und PDM-Systeme, sowie die Schnittstellen zwischen diesen entwickelt. Der Auslöser für eine Entwicklung ist hier der Kunde, der mit dem Vertrieb Kontakt aufnimmt.

Projektabwicklung

Der Vertrieb leitet diese Anfrage an einen Projektmanager weiter, der zur Aufwandsabschätzung gegebenenfalls eine Grobspezifikation formuliert. Wird der Auftrag ausgelöst, erarbeitet ein Entwickler eine Feinspezifikation, die von dem Projektleiter anschließend überprüft wird. Da hier nur kleine Projekte bearbeitet werden, ist immer nur ein Entwickler mit einem Projekt betraut, so dass jeglicher Koordinationsaufwand entfällt.

Eingesetzte Tools

Als Entwicklungswerkzeug wird im wesentlichen Visual C++ von Microsoft verwendet. Für die Verwaltung der verschiedenen entwickelten Softwareversionen und für die Revisionen wird das PDM-System SmarTeam eingesetzt. In diesem System wird der dem Projekt

zugeordnete Ordner jeweils ein- und ausgecheckt. Auf diese Weise kann jeder Entwicklungsstand einer Software jederzeit wiederhergestellt werden.

Qualitätssicherung

Für eine spezielle Entwicklung wird ein Testszenario entworfen, welches mit dem Kunden abgesprochen wird. Dieses Testszenario lehnt sich dabei an das vom Kunden erstellte, bzw. durch den Kunden abgezeichnete Pflichtenheft an. Die Verteilung der Software erfolgt mittels Updates über das Internet. Die Software wird durch den Support getestet, nicht durch den Entwickler. Für das Testszenario wird ein Testprotokoll erstellt.

Dokumentation

Für die Form der Dokumentation der Softwareentwicklung existiert keinerlei Vorschrift, sie bleibt im Ermessen des Entwicklers. Vom Entwickler, bzw. dem Tester der Software wird ein Handbuch erstellt und in SmarTeam abgelegt.

Wartung und Weiterentwicklung

Die Wartung und die Weiterentwicklung des Software-Produktes werden über Supportcalls durch den Vertrieb veranlasst. Der Kunde erhält nicht den Quelltext sondern nur die kompilierte Software und gegebenenfalls das Benutzerhandbuch.

Kundeninformationen

Die Kundenanfragen werden mit Hilfe einer speziell vom Dienstleister entwickelten Software über das Internet gestellt, im PDM-System verwaltet und automatisch immer an den Support weitergeleitet. Dadurch dass der Support über einen Zugang zu dem PDM-System verfügt, ist er über alle zu dem Projekt abgelegten Informationen im Bilde und kann somit mit der für ihn erforderlichen Sicherheit gegenüber dem Kunden argumentieren.

2.2.2 Softwareentwicklung im PDM-Umfeld

Dienstleistungsspektrum

Bei diesem Dienstleister werden mit ca. 10 Mitarbeitern im wesentlichen zwei Bereiche abgedeckt. In erster Linie wird das eigene PDM-System Profi-DB weiterentwickelt und

daneben wird für das PDM-System Workmanager von CoCreate die Benutzeroberfläche teilweise entwickelt. Für die Eigenentwicklung Profi-DB werden auch kundenspezifische Anpassungen vorgenommen, um das System bei Kunden in eine bestehende Architektur zu integrieren, was teilweise umfangreiche Schnittstellen-Entwicklungen erfordert. Zusätzlich werden noch Makro-Anpassungen für das CAD-System ME10 vorgenommen.

Projektabwicklung

Liegt ein Auftrag vor, so werden zunächst die zu erstellenden Funktionalitäten aufgelistet und mit Hilfe von Erfahrungswerten eine Aufwandsabschätzung durchgeführt. Mit Auftragsbeginn werden die Phasen Spezifikation, Entwicklung, Test und Auslieferung durchlaufen Hierbei werden keine speziellen Werkzeuge zur Unterstützung der Entscheidungsprozesse eingesetzt. Das Projektmanagement realisiert ein Projektmanager. Die Koordination der Entwickler erfolgt über eine Modularisierung der Aufgaben und über eine feine Dateistruktur, so dass sich nur wenig Überschneidungen der Aufgaben ergeben.

Eingesetzte Tools

Für das Versions- und Revisionsmanagement wird Visual Source-Safe von Microsoft eingesetzt. Damit kann jeder gespeicherte Zwischenstand der Entwicklungen wieder hergestellt werden. Zusätzlich erstellte Dokumentationen werden im Visual Source-Safe verwaltet.

Qualitätssicherung

Das Testen erfolgt in der Regel ohne spezielle Vorgaben anhand der Spezifikation der Software unter Berücksichtigung der während der Entwicklung ausgeführten Änderungen. Der eigentliche Test erfolgt durch den Support, bzw. durch Reseller. Für das Testen kundenspezifischer Software wird ein Testszenario auf der Basis der Kundendaten erstellt. Bei der für CoCreate erstellten Software (Workmanager Oberfläche) erfolgt das endgültige Testen durch CoCreate selber. Der Kunde ist bei der Erstellung des Pflichtenheftes und an der Abnahme jeweils beteiligt. Die Abnahme erfolgt beim Kunden durch einen Test anhand des Pflichtenheftes und der durch den Kunden bereitgestellten Testdaten

Dokumentation

Die Dokumentation der Software erfolgt im wesentlichen durch den Entwickler mit Hilfe von Inline-Kommentaren[3] in den Quelltexten. Die für die Erstellung der Software verwendeten Kundenumgebungen werden komplett gespeichert, so dass sie bei Problemen jederzeit wieder aktiviert werden können.

Wartung und Weiterentwicklung

Der Anstoß für eine Weiterentwicklung der Standardsoftware geht meistens von der Notwendigkeit aus, sich den neuen Technologien anzupassen. Als Beispiel seien hier genannt, die sich ändernden Betriebsystemversionen (z.B. 2000->XP), neue Entwicklungswerkzeuge (MSVC „.NET“) oder neue Versionen der Integrierten Software (CAD-Systeme usw.). Daneben haben aber auch aktuelle Markttrends und Kundenwünsche einen erheblichen Einfluss auf die Weiterentwicklung. Markttrends werden allerdings erst umgesetzt, wenn es sich abzeichnet, dass ein Markttrend sich auch durchsetzt. Allerdings muss vor der Software-Anpassung noch der Aufwand gegen den Kundennutzen abgewogen werden.

Der Support besitzt einen Zugriff auf sämtliche Quelltexte und auf sämtliche Kundenumgebungen und kann diese gegebenenfalls zur Überprüfung und Fehlerbeseitigung jederzeit wieder aufsetzen. Probleme, die aus Änderungen des Kunden an den Quelltexten resultieren, werden nicht supportet bzw. nur gegen Extragebühren.

Kundeninformationen

Der Kunde erhält alle Quelltexte und sämtliche Dokumentation über die Software.

2.2.3 Softwareentwicklung im SAP-Umfeld

Dienstleistungsspektrum

Die IT-Abteilung mit ca. 1000 Mitarbeitern des hier analysierten Konzern-Unternehmens ist im wesentlichen interner Dienstleister, der die notwendigen Anpassungen an das im Konzern

[3] Kommentare zur Beschreibung des Algorithmus innerhalb des Quelltextes

vorhandene SAP-R3-System auf der Basis der Programmiersprache ABAP durchführt. Des weiteren werden noch spezielle Dienstleistungen in Assembler und in COBOL [7] für noch laufende PPS/PDM-Systeme erbracht. Die Spanne reicht von kleinen Anpassungen (<1200€) bis hin zu großen Projekten, die bis zu 100 Mitarbeiter beschäftigen.

Der Kunde tritt an einen so genannten Customer Liaison Manager heran, der den Kontakt zu den Vertriebsmitarbeitern herstellt. In der täglichen Praxis wird häufig ein Vertriebsmitarbeiter direkt angesprochen, der dann den Liaison Manager informiert.

Projektabwicklung

Den Vertriebsmitarbeitern stehen Stäbe zur Seite, die den Aufwand und den Zeitrahmen für das Projekt abschätzen. Dabei werden sie gegebenenfalls von externen Beratungsfirmen unterstützt, welche die Aufwandsabschätzung, Spezifikation und Definition des Softwareprojektes vornehmen. Das technische Design wird mit Unterstützung der Entwickler erstellt. Je nach Größe des Projektes, wird ein Projektleiter eingesetzt, der die Aufgaben an die einzelnen Entwickler verteilt und den Entwicklungsfortschritt kontrolliert. Wenn Entwickler an Aufgaben arbeiten, die voneinander abhängig sind, so müssen sich diese absprechen. Das Projektmanagement erfolgt über einen Projektmanager, der die Aufgaben an die einzelnen Entwickler verteilt und überwacht, sowie den Kontakt mit den externen Dienstleistern und dem Liaison Manager herstellt.

Die Anpassung der SAP-R3-Software erfolgt im wesentlichen im SAP-System selbst. Dazu steht im SAP-R3-System eine eigene Entwicklungsumgebung, die so genannte Workbench zur Verfügung. Dort wird in der Transportschicht festgelegt, in welches Teil-System die Software eingespeist wird. Unterhalb der Transportschicht werden die einzelnen Projekte angelegt. Den Projekten werden Aufgaben zugeordnet (die Teilschritte bei der Softwareentwicklung). Den Aufgaben werden Mitarbeiter zugeordnet, denen die Quelltexte der Software zugeordnet werden. Die Versionsverwaltung erfolgt auf der Ebene der Aufgaben. Die Freigabe erfolgt von unten nach oben. Ein Entwickler gibt seine Quellen frei. Sobald alle Quellen einer Aufgabe freigegeben sind, kann die Aufgabe freigegeben werden. Sind alle Aufgaben freigegeben, so kann das Projekt freigegeben werden. Über die Transportschicht wird die fertige Software dann in ein Testsystem und anschließend in das

Produktivsystem überspielt. Insgesamt existiert in einem Projekt für die einzelnen Projektschritte eine 16-stufige Statusänderung.

Eingesetzte Tools

Sämtliche Dateien und alle Aufgaben werden in den verschiedenen SAP-R3-Systemen abgelegt, die auch für den Workflow verantwortlich sind. Ansonsten werden hier keine speziellen CASE-Tools eingesetzt. So erfolgt z.B. die Spezifikation und Definition über einfache EXCEL-Tabellen. Welche Tools von den externen Dienstleistern eingesetzt werden, ist nicht bekannt.

Das Versions- und Revisionsmanagement erfolgt ebenfalls innerhalb der SAP-Workbench. Jeder Entwicklungsstand einer Software kann wiederhergestellt werden, wobei jedoch keine Verzweigungsbäume existieren und Abhängigkeiten verschiedener Versionen voneinander nicht nachvollzogen werden können. Änderungen und Änderungsanfragen an bzw. zu der SAP-Software (z.B. spezielle Reports) werden in einer Datenbank eingetragen, dem sogenannten Change Request Server.

Qualitätssicherung

Vor Abschluss eines Projektes erfolgt ein sogenannter Integrationstest, bei dem die entwickelte Software in einer Umgebung getestet wird, in der bereits alle Komponenten der späteren Produktivumgebung vorhanden sind. Der Termin für diesen Integrationstest steht bereits zu Beginn des Projektes fest, kann aber gegebenenfalls verschoben werden. Der Integrationstest findet in einem ausreichenden Zeitfenster bis zur Produktivschaltung beim Kunden statt. Getestet wird mit einem Testszenario, welches im Laufe der Entwicklung erarbeitet wird. Getestet wird durch die Entwickler und auch durch Mitarbeiter aus den jeweiligen Fachabteilungen (Kunde). Der Kunde nimmt an diesem Integrationstest teil.

Nach dem Integrationstest und abschließender Fehlerbeseitigung wird die Software beim Kunden installiert und produktiv geschaltet. Dies geschieht in der Regel an einem Wochenende oder über die Betriebsferien (z.B. Weihnachten). Nach dem Integrationstest sollte die Software soweit funktionieren, dass keine weiteren Korrekturen mehr notwendig sind. Sollte dies dennoch der Fall sein, so liegt dies in der Regel an unzureichenden

Informationen durch den Kunden bzw. an nicht vollständigen oder fehlerhaften Testszenarien während des Integrationstests. Nach der Produktivschaltung wird das Entwicklungsteam gegebenenfalls noch für einen kurzen Zeitabschnitt zusammengehalten, um auftretende Probleme am Produktivsystem schnell zu lösen.

Dokumentation

Die Dokumentation des Quellcodes erfolgt über Inline-Kommentare und gegebenenfalls über zu der Software zugelinkte Dokumente. Diese werden jedoch nicht vom SAP-System verwaltet, sondern auf speziellen Freigaben abgelegt, auf denen auch der in der Entwicklung befindliche Quellcode abgelegt ist. Eine gesonderte Dokumentation der Software erfolgt in der Regel nicht (Handbücher usw.), jedoch werden bei größeren Projekten Schulungsunterlagen erstellt, mit denen die Mitarbeiter des Kunden geschult werden. Die Dokumentation unterliegt keinem Versionsmanagement.

Wartung und Weiterentwicklung

Die Wartung und die Weiterentwicklung wird durch den Kunden veranlasst, gegebenenfalls auch durch die Entwickler, wenn diese in ihrem Tagesgeschäft auf ein Problem stoßen. Bei kleinen Änderungen haben die Entwickler die Freiheit diese selbst unmittelbar vorzunehmen. Bei größeren Problemen muss eine Abwägung von Kosten/Nutzen gemacht werden und für eine Mittelzuweisung muss ein offizielles Projekt generiert werden. Der Support hat jederzeit Zugriff auf das SAP-R3-System, auf den Change Request Server und auf alle Freigaben auf denen die Software abgelegt ist.

Kundeninformationen

Der Kunde erhält nur die fertige Software und gegebenenfalls die Schulungsunterlagen. Alle anderen Informationen bleiben bei dem Dienstleister.

2.2.4 Softwareentwicklung im PLM-Umfeld

Dienstleistungsspektrum

Der hier analysierte Dienstleister mit 200 Mitarbeitern stellt im wesentlichen Customizing-Software im Zusammenhang mit verschiedenen Anwendungen (CAD-Programme / PDM-Systeme / SAP / IMAN / Schnittstellen) her. Die Kundenanfrage geht an den Vertrieb.

Projektabwicklung

Der Vertrieb leitet die Anfrage an einen Projektmanager, welcher die wirtschaftliche und technische Verantwortung trägt, da dieser die notwendige Programmiererfahrung besitzt. Dieser erstellt das Pflichtenheft und die Grob- sowie Feinspezifikation. Ebenso wird durch ihn die Aufwandsabschätzung durchgeführt. Die Projektingenieure (Programmierer, Techniker und Supportmitarbeiter) werden durch eine genaue Projektspezifikation koordiniert, d.h. jeder Entwickler weiß genau, welche Aufgabe er hat. Bei sich überschneidenden Aufgaben sprechen die Entwickler sich gegebenenfalls ab. Die Entwicklung der Software erfolgt durch die Projektingenieuren unter Aufsicht des Projektmanagers.

Eingesetzte Tools

Die Projektpläne werden mit MS-Project erstellt. Darüber hinaus liegen bestimmte Dokumente als Vorlagen bereit, wie z.B. Vorlagen für das Pflichtenheft. Das Versions- und Revisionsmanagement wird über die GNU-Software RCS (Revision Control System) realisiert. Bei einigen C++/Autocad-Projekten wird aber auch MS-Source-Safe verwendet. Mit RCS kann jeder beliebige Stand der Software wiederhergestellt werden. Für jedes Projekt wird über ein spezielles Tool, welches vom Dienstleister selbst entwickelt wurde, ein spezieller genormter Verzeichnisbaum erstellt.

Qualitätssicherung

Nach Fertigstellung geht die Software an den Support, welcher in die Software eingewiesen wird und diese auch testet. Ist der Test erfolgreich, so wird die Software an den Kunden ausgeliefert und dort in einen Testbetrieb überführt. Der Kunde testet die Software anhand

seiner Daten. Ist die Testphase erfolgreich verlaufen, so erfolgt auf der Basis des Pflichtenheftes die Abnahme durch den Kunden. Danach wird die Software produktiv geschaltet und gegebenenfalls dem Support und der Wartung zugeführt. Die Qualitätssicherung erfolgt in der Entwicklungsphase durch den Projektmanager, der die Entwickler überwacht, damit diese auch einen sauberen Code mit ausreichenden Kommentaren abliefern. Der Kunde ist in die Projektarbeit über das Pflichtenheft und durch die der Entwicklung nachfolgende Testphase mit einbezogen.

Die Abnahme durch den Kunden erfolgt nach einer Testphase auf einem Testsystem anhand des Pflichtenheftes. Mit dem RCS können nach entsprechender Konfiguration automatisch Installationsversionen der Software erstellt werden. Daneben existieren auch spezielle RCS-basierte Tools um die Erstellung und Verwaltung so genannter Patches, die dazu dienen Fehlerbereinigung in einer installierten Kundenumgebung durchzuführen.

Dokumentation

Der Quellcode wird von den Entwicklern durch Inline-Kommentare dokumentiert. Weiterhin werden für die Software je nach Projektumfang System- und Benutzerhandbücher durch den Projektmanager erstellt. Die Dokumentation unterliegt keinem Versionsmanagement. Ein solches könnte aber prinzipiell auch mit Hilfe des RCS realisiert werden. Die Dokumentation wird im Projektverzeichnis abgelegt.

Wartung und Weiterentwicklung

Die Wartung und die Weiterentwicklung wird im wesentlichen über Kundenanfragen veranlasst. Der Support hat Zugriff auf alle Informationen über die Software.

Kundeninformationen

Der Kunde erhält je nach Vertrag die fertige Software plus Dokumentation oder gegebenenfalls auch den gesamten Quellcode, welcher dann nicht mehr Eigentum des Dienstleisters ist und auch in weiteren Projekten mit anderen Kunden nicht verwendet werden kann.

2.3 Beurteilung der Defizite der analysierten Praxisbeispiele

Bei der Betrachtung der Analyse fällt auf den ersten Blick auf, dass die einzelnen Szenarien recht unterschiedlich sind und in den meisten Fällen kein roter Faden in der Softwareentwicklung sichtbar wird. In einer kurzen Zusammenfassung stellen sich die einzelnen Bereiche wie folgt dar.

2.3.1 Projektabwicklung

Eine Softwareentwicklung unterliegt im wesentlichen zwei großen Bereichen, dem Daten- und dem Prozessmanagement. Für ein optimales Zusammenwirken dieser beiden Bereiche ist ein Projektmanagement notwendig, welches jederzeit auf die aktuellen Daten und den Stand des Prozessfortschritts zugreifen kann. Diese Notwendigkeit hat keiner der analysierten Dienstleister für den gesamten Softwarelebenszyklus realisiert. Zwei der Dienstleister haben dafür weniger geeignete Systeme eingesetzt und die fehlenden Funktionalitäten wie die anderen Dienstleister mit Hilfe jeweils selbst erstellter Dateiverzeichnisse ergänzt, um die Softwareentwicklung weitestgehend störungsfrei abzuwickeln.

Dieses kann natürlich nicht immer gelingen, da bei dieser Vorgehensweise der einzelne Beteiligte nicht durch einen übergeordneten Mechanismus geführt wird, sondern sein Handeln ausschließlich durch eine erhöhte Eigeninitiative ausgelöst werden muss. Diese Handlungsweise erschwert erheblich die Koordination der Projektvorgaben und verlangt nach zusätzlichen Kontrollmechanismen. Des weiteren ist der einzelne Entwickler nicht unbedingt in das jeweilige Gesamtziel mit eingebunden, was die Motivation des Einzelnen nicht unbedingt fördert.

2.3.2 Eingesetzte Tools

Als Tools zur Verwaltung der verschiedenen Softwareprojekte werden bei den Dienstleistern die Systeme 1. MS-SourceSafe, 2. SmarTeam, 3. SAP-Workbench, 4. Revision Control System von GNU und 5. Freigabe als einfache Ablage in einem Verzeichnis eingesetzt. Die Eigenschaften dieser Systeme lassen sich folgendermassen charakterisieren:

- **MS-SourceSafe** wurde von Microsoft zur Versions- und Revisionsverwaltung in Softwareprojekten entwickelt. Naturgegeben verfügt dieses System über eine gute Integration in Microsoft Produkte. Die Ablage der Versionen erfolgt in einer Datenbank, so dass sich jeder beliebige Projektstand wiederherstellen lässt. Dateien lassen sich ein- und auschecken und es erfolgt auch eine Sperrung ausgecheckter Dokumente. Es lassen sich jedoch keine Beziehungen zwischen verschiedenen Dokumenten herstellen. Ebenso fehlt eine Benutzerverwaltung mit der Möglichkeit Zugriffsrechte individuell zu steuern, und ein Workflow, welcher eine Steuerung oder Überwachung des Entwicklungsprozesses ermöglicht.

- **SmarTeam** ist ein für produzierende Unternehmen entworfenes PDM-System, welches im wesentlichen für den Einsatz in einem CAD/CAM-Umfeld gedacht ist. Es bietet daher keine Integration in eine Entwicklungsumgebung. Die Ablage der Dateien erfolgt in einem so genannten Vault (Tresor) und die Verwaltung wird mittels einer relationalen Datenbank realisiert. Das System verfügt über ein Dokumentenmanagement und ein Workflow-Modul, welches jedoch in dem untersuchten Unternehmen im Rahmen der Softwareentwicklung nicht eingesetzt wurde.

.

- **SAP-Workbench** ist eine sehr komfortable Entwicklungsumgebung mit vollständig integrierter Dokumentenverwaltung über das SAP-System, welche viele Belange der Softwareentwicklung abdeckt. Allerdings ist es nur für die Entwicklung von ABAP-Skripten im Rahmen eines SAP-Systems entwickelt worden, und kann nicht für anderweitige Problemstellungen oder andere Programmiersprachen verwendet werden.

- **RCS** (Revision Control System) ist eine freie Software unter der so genannten GNU Public License. Sie stellt ein sehr mächtiges Konfigurationsmanagement dar und ist

speziell für die Ansprüche in der Softwareentwicklung konzipiert. Allerdings handelt es sich hier um ein reines Kommandozeilen-Tool, welches im wesentlichen unter UNIX seine Anwendung findet. Es gibt auch Anpassungen an DOS oder z.B. RiscOS (Acorn). Eine besondere Eigenschaft dieser Software ist, dass sich die automatische Erstellung von Installationsversionen einer entwickelten Software konfigurieren lässt.

- **Freigabe** ist ein freigegebenes Verzeichnis, auf welches über ein Netzwerk zugegriffen werden kann. Dort können Dokumente abgelegt und sortiert werden. Weitere Eigenschaften existieren nicht. Alle Tätigkeiten der Projektabwicklung basieren auf der strukturierten Ablage der Dateien in der Freigabe.

Des weiteren wurden die hier eingesetzten Systeme hinsichtlich bestimmter allgemeingültiger Anforderungen gemäß Tabelle 2.1 untersucht. Die einzelnen Kriterien sind dabei in der Reihenfolge ihrer Wichtigkeit von oben nach unten sortiert.

Anforderung	Syst. 1	Syst. 2	Syst. 3	Syst. 4	Syst. 5
Beliebige Programmiersprachen	o	x	-	x	x
Ablage von Dokumenten	x	x	x	x	x
Versions- und Revisionsverwaltung	x	x	x	x	o
Freigabeschemata	x	x	x	x	-
Verfolgen von Abhängigkeiten	-	x	-	-	-
Sperren von Dokumenten	x	x	x	x	-
Workflow	-	x	x	-	-
Änderungsverfolgung	x	-	x	x	-
Grafisches Front-End	x	x	x	-	-
API	x	x	x	-	-
Benutzerverwaltung	-	x	x	-	-
Verlinkung beliebiger Dokumente	-	x	-	-	-
Erstellung von Installationsversionen	-	-	x	x	-
Web Interface	-	x	x	-	-
Einsatzplanung der Entwickler	-	-	x	-	-

Tabelle 2.1: Vergleich der eingesetzten Tools

Nach der Analyse ist die SAP-Workbench ein gut geeignetes Werkzeug zur Softwareentwicklung, jedoch ist die Workbench vollständig in das SAP-R3-System integriert

und nur auf die Entwicklung von SAP-ABAP-Scripten [8] beschränkt, was ein absolutes KO-Kriterium für die Anforderungen darstellt. Andere Programmiersprachen als ABAP werden nicht unterstützt und können auch nicht nachträglich integriert werden. Ein anderes entscheidendes Kriterium ist das Verfolgen von Abhängigkeiten, d.h. das Feststellen, welche Version einer Software mit welchen anderen Versionen zusammenarbeitet und welche nicht.

2.3.3 Qualitätssicherung

Die Analyse hat gezeigt, dass insbesondere die Qualitätssicherung von jedem Dienstleister unterschiedlich gehandhabt wird, obwohl letztlich jeder Anwender im praktischen Einsatz des jeweiligen Software-Produktes die gleichen Probleme haben wird. Das Fehlen der Informationen bezüglich des Entwicklungsvorganges der Software erschwert den praktischen Einsatz immer dann, wenn hier jeweils nur Teilsysteme entwickelt und zu einem komplexen heterogenen Softwaresystem zusammengeführt wurden. Dies gilt um so mehr, wenn die Teilsysteme ursprünglich nichts miteinander zu tun hatten, aber aus Gründen der Rationalisierung miteinander integriert wurden.

Eine Ursache liegt darin begründet, dass in diesen Situationen häufig sehr verschiedene Programmiersprachen und Entwicklungsumgebungen zum Einsatz kommen. Ähnliche Probleme treten auf, wenn das Software-Produkt von der zur Entwicklung benutzten Betriebssystem- bzw. Hardware-Plattform auf eine andere Plattform portiert wird, ohne dass die Architektur der Software ausreichend angepasst wird. Gerade dann ist das Vorhandensein der sogenannten „Entwicklungsdokumentation“ besonders wichtig.

2.3.4 Dokumentation und Kundeninformationen

Die meisten Softwareentwickler betrachten die Erstellung einer Dokumentation, häufig nur bestehend aus einem Anwender- und einem Administrator-Handbuch, als notwendiges Übel, welches zum Verkauf der Software einfach dazu gehört. Aus diesem Grunde wird diese Arbeit auch immer erst nach Abschluss der eigentlichen Entwicklung ausgeführt, der sogenannten Codierung. Dies führt dazu, dass häufig wichtige Details, die bei der

Entwicklung entdeckt wurden, später einfach keine Berücksichtigung mehr finden. Gerade diese scheinbar unwichtigen Details aber können dem späteren Anwender das Leben häufig erleichtern. Hier fehlt es seitens des Projektmanagements an den erforderlichen Vorgaben für die einzelnen Entwickler bezüglich einer hinreichend ausgereiften Dokumentation.

Für das Arbeiten mit dem Software-Produkt sind diese Handbücher sicherlich ausreichend. Für die Installation und die spätere Weiterentwicklung muss gegebenenfalls auch der Quellcode in irgendeiner gut dokumentierten Form vorliegen. Sogenannte Inline-Kommentare allein reichen nicht aus, um zu einem späteren Zeitpunkt die Software entsprechend den neuen Gegebenheiten des praktischen Alltags anzupassen. Diese häufig „gewachsenen“ schlecht dokumentierten Systeme können nur unter äußerst schwierigen Bedingungen überarbeitet werden, da wegen der mangelnden Vorgaben die zur Lösung führenden Algorithmen in der Regel nur unzureichend beschrieben worden sind. Darüber hinaus sind die ehemaligen Entwickler häufig auch nicht mehr verfügbar.

2.3.5 Wartung und Weiterentwicklung

Für eine gute Wartung und Weiterentwicklung muss gewährleistet sein, dass alle während der Entwicklung gewonnenen Informationen auch später verfügbar sind. Dies ist in der Regel nicht der Fall, da der einzelnen Mitarbeiter nicht in ein ganzheitliches Konzept eingebunden ist und somit nur seine Teilaufgabe sieht. Dies führt dazu, dass er selbst entscheidet, welche seiner gewonnenen Informationen für den Softwarelebenszyklus wichtig sind. Da aber in den meisten Fällen die Entwicklung und der spätere Support von verschiedenen Personengruppen ausgeführt werden, muss diese Vorgehensweise zwangsläufig nicht das Optimum für den Anwender erzielen.

Dies zeigt doch recht deutlich, dass in der heutigen Softwareentwicklung die nachfolgend geschalteten Bereiche Wartung und Weiterentwicklung nicht mit der nötigen Sorgfalt berücksichtigt werden, d.h. die Wartung und Pflege der Software ist **nicht** bis zu deren Ablösung sichergestellt. Dieser gravierende Nachteil führt in der industriellen Produktion häufig zu erheblichen Produktionsausfällen und damit zu enormen Kosten. Auch diese Schwachstelle gilt es zukünftig zu verhindern.

2.4 Zusammenfassung der momentanen Defizite in der Softwareentwicklung

Bevor nun die Anforderungen an ein Konzept für einen ganzheitlichen Ansatz im Hinblick auf einen Softwarelebenszyklus aufgestellt werden, erscheint es äußerst sinnvoll, die momentan vorhandenen Defizite in einer kurzen Zusammenfassung darzustellen. Hierbei werden nicht nur die Defizite der Analyse genannt sondern darüber hinaus auch die Defizite die der Verfasser in seiner langen Tätig im Bereich der Softwareentwicklung selbst erfahren hat bzw. in zahlreichen Begegnungen mit anderen Entwicklern auf Seminaren und User-Treffs ausführlich diskutiert hat.

Für viele verschiedene Programmiersprachen existieren fortgeschrittene Entwicklungs-werkzeuge, die dem Programmierer vielfältige Werkzeuge zur Gestaltung, Analyse und Fehlersuche bei der Softwareentwicklung bereitstellen. Teilweise gibt es sogar Hilfsprogramme zur automatischen Erstellung von Standardcode. Zu diesen Entwicklungsumgebungen im Umfeld von WINDOWS zählen z.B. Microsoft Visual Studio, Borland Delphi oder IBM Visual Age. Diese Entwicklungsumgebungen basieren darauf, dass alle für die Entwicklung relevanten Daten lokal, oder zumindest auf einem freigegebenen Verzeichnis auf einem Server-Rechner exklusiv dieser Entwicklungsumgebung zur Verfügung stehen.

Dies ist unproblematisch, solange ein Entwickler alleine an der Programmierung arbeitet. Mit den steigenden Anforderungen an die Komplexität der Software arbeiten jedoch mehrere Entwickler gleichzeitig an einer Software-Entwicklung (vgl. Abb. 2.2), was wiederum häufig zu Problemen führt. Diese Probleme haben verschiedene Ursachen. Da die einzelnen Entwickler innerhalb ihrer Entwicklungsumgebung immer nur jeweils lokal auf ihren Rechnern arbeiten, verlangt eine übergeordnete Stelle eine jederzeitige lokale Kopie aller Dateien des Softwareprojektes. Dies führt zu einer Datenredundanz und mit der Zeit zu divergierenden Entwicklungsständen der einzelnen Entwickler. Somit müssen innerhalb von vorgegebenen Zeitintervallen die einzelnen Entwicklungsstände zu einer konsistenten Entwicklungsstufe des Gesamtprojektes zusammengeführt werden, damit die einzelnen Entwickler ihre Arbeit auf einer gemeinsamen Basis fortsetzen können.

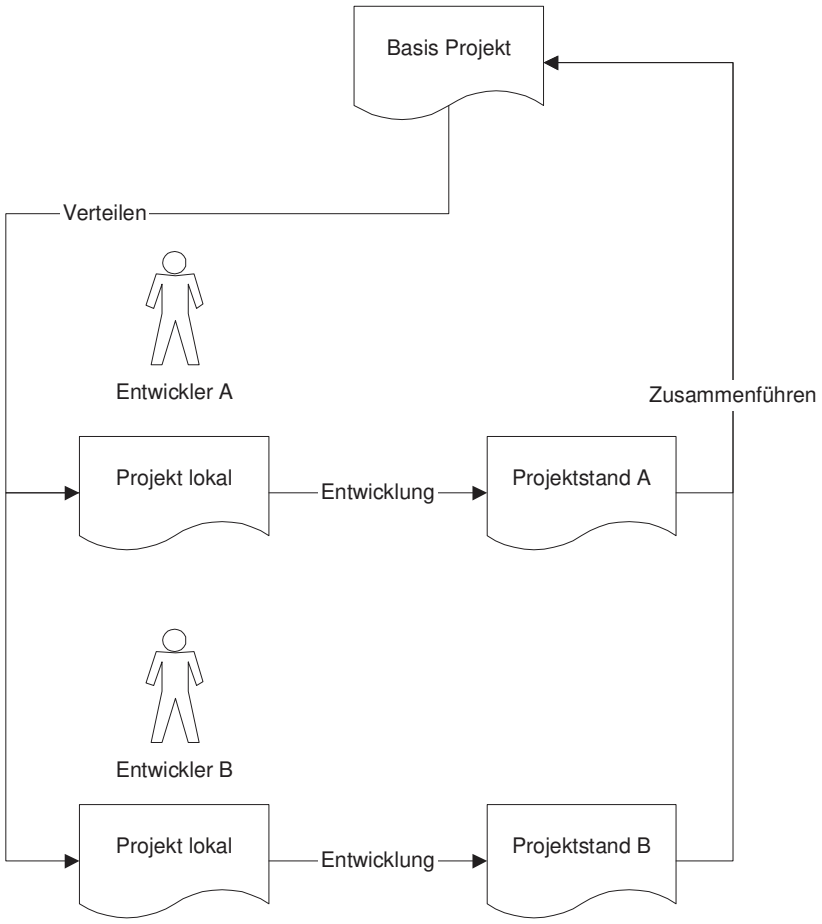


Abbildung 2.2: Zusammenspiel von Entwickler mit Projekt

Diese Problemstellung ist die gleiche Problemstellung, die auch im Bereich der „herkömmlichen“ Produktentwicklung bekannt ist. Im PLM-Umfeld findet diese Tatsache unter dem Begriff Concurrent- oder Simultaneous Engineering Berücksichtigung. Hier wurde diese Problemstellung bereits erkannt und es existieren verschiedene Mechanismen um eine gleichzeitige Bearbeitung eines Projektes durch mehrere Mitarbeiter zu ermöglichen.

- Bei den Software-Produkten handelt es sich nicht um ein einziges Computerprogramm, welches eigenständig ausgeführt wird, sondern um ein komplexes System aus verschiedenen Modulen, welche oft auch mit unterschiedlichen Programmiersprachen codiert sind. Eine einheitliche Verwaltung dieser unterschiedlichen Module ist in der Regel nicht gewährleistet.
- Es hat sich gezeigt, dass die „Lebenserwartung“ von Software deutlich höher ist, als ursprünglich angenommen. Ein häufig auftretendes Problem besteht darin, dass

Computerprogramme, deren Entwicklung oft 20 Jahre zurückliegt, weiterentwickelt oder gewartet werden müssen. In der Regel handelt es sich dabei um gewachsene Systeme, die meist nur unzureichend dokumentiert sind.

- Die Systeme sind auf einer Betriebsystem- bzw. Hardware-Plattform entstanden und werden dann ohne ausreichende Anpassung der Architektur auf andere Plattformen portiert.
- Die Software wurde entwickelt ohne dass in ausreichendem Maße die Wartung und Pflege der Software bis zu deren Ablösung berücksichtigt wurde.

Die hier durchgeführte Analyse hat neben den vorab aufgeführten Defiziten in den einzelnen Problemfeldern außerdem gezeigt, dass mit einer gezielten Vorgehensweise während der Entwicklung, die auch die Zeit danach berücksichtigt, erhebliche Kosten eingespart werden können. Folgende Faktoren beeinflussen die Kosten des Softwarelebenszyklus während und nach der eigentlichen Entwicklung wesentlich:

- Je größer das Software-Entwicklungsprojekt ist, um so mehr Aufmerksamkeit muss den frühen Phasen der Softwareentwicklung gewidmet werden. Übereinstimmend haben alle Befragten ausgesagt, dass die Fehler der Entwicklung mit wachsender Projektgröße später äußerst schwierig und nur mit großem finanziellen Aufwand zu korrigieren sind.

- Die meisten der während einer Softwareentwicklung entstehenden Dokumente, sind keine Quellcode-Dokumente, sondern Dokumente, welche die Software auf andere Art beschreiben. Hierzu gehören insbesondere solche Dokumente, wie Pflichten-/Lastenheft, Tabellen mit Funktionalitätslisten, Diagramme, Protokolle, Fehlerlisten, Änderungslisten, UML-Dokumente[4] usw. . Die Erstellung und der Umgang mit diesen Dokumenten würde bei einer sachgemäßen Behandlung erhebliche Kosten einsparen. Denn nicht nur während der Entwicklungsphase kommt es darauf an, diese Dokumente effizient zu verwalten und zu speichern, sondern auch während der sich anschließenden Wartungsphase, um einen schnellen Zugriff auf die für die Wartung notwendigen Informationen zu haben.

[4] Unified Modeling Language

- Die größten Kosten während des Softwarelebenszyklus werden nicht in der eigentlichen Entwicklungsphase sondern in der Wartungsphase verursacht. Generell kann hier gesagt werden, dass 2/3 – 4/5 der Kosten einer Software in der Wartungsphase entstehen [9]. Dies hängt auch damit zusammen, dass die mit der Entwicklung der Software betrauten Personen später nicht auch für die Wartung verantwortlich sind, so dass diese um so mehr auf eine genaue und vollständige Dokumentation und auf einen schnellen Zugriff auf die notwendigen Informationen angewiesen sind, um eventuelle Probleme schnell und kostengünstig zu beseitigen. Diese Sichtweise wird momentan während der Entwicklung total vernachlässigt.

    - Für die Beurteilung der Weiterentwicklungskosten muss gemäß den folgenden zwei unterschiedliche Arten von Software-Produkten unterschieden werden:

1. **Allgemeingültige Software**, die für einen großen Kundenkreis generelle Funktionalitäten bereitstellen. Zu dieser Art Software gehören z.B. Betriebssysteme, Textverarbeitungsprogramme, Spiele oder ein PC-BIOS[5].
2. **Kundenspezifische Software**, die für einen speziellen Kunden nach dessen Vorgaben erstellt wird. Hierzu gehören spezielle Anwendungsprogramme, Anpassungen an allgemeingültige Software, aber auch Schnittstellen zwischen verschiedenen Software-Produkten.

Im ersten Fall werden die Kosten für die Weiterentwicklung vom Hersteller getragen, da dieser die Software der ständig fortschreitenden Entwicklung am Markt anpassen muss. Oft können diese Kosten über kostenpflichtige Updates der Software auf den Kunden abgewälzt werden, insbesondere dann, wenn die Wünsche für die Anpassung beim Kunden oder bei einem Dritten (z.B. neue Version des Betriebssystems) vorliegen. In jedem Fall aber muss der Softwarehersteller in Vorleistung gehen.

Im zweiten Fall liegen die Kosten im wesentlichen beim Anwender, da dieser die Software speziell beim Hersteller gekauft hat und nach der Abnahme der Software die Verantwortung dafür in der Regel auf den Kunden geht. Wenn der Hersteller der Software trotz Kauf die

[5] Basic Input Output System

Verantwortung übernehmen soll, werden für den Kunden kostenpflichtige Wartungsverträge abgeschlossen.

Die hier dargestellten Defizite lassen sich nur mit einem Software-Lifecycle-Managementsystem (SLM) lösen. Zweck des SLM ist es jedoch nicht, Fehler in den Planungsphasen zu verhindern, insbesondere Fehler, die durch mangelndes Verständnis der Problemstellung entstehen. Dies ist Aufgabe der Projektleiter und Softwarespezialisten. Ziel ist es vielmehr, mit diesen Fehlern schnell und effizient umgehen zu können und alle getroffenen Entscheidungen nachvollziehbar zu machen. Im Folgenden werden nun die Anforderungen an ein SLM formuliert.

3 Forderungen an ein Software-Lifecycle-Managementsystem

Die gängige Vorstellung über den Softwareentwicklungsprozess ist sicherlich, dass es irgendwo ein Idee für eine Software gibt und ein Programmierer setzt diesen Wunsch in seinem stillen Kämmerlein nach bestem Wissen und Vermögen um, oder bei größeren Projekten setzen sich spontan mehrere Programmierer vor die Rechner und fangen an wie wild zu programmieren. Diese Vorgehensweise der Softwareentwicklung würde bezüglich des neuen Produktes im wesentlichen immer eine komplette Neuentwicklung darstellen, die dementsprechend auch meistens sehr teuer wird. Zudem ist die so entstandene Software in den meisten Fällen ein Produkt ihrer Schöpfer und kann auch nur von diesen verstanden werden. Um dieses Problem in den Griff zu bekommen, wurde schließlich der objektorientierte Softwareentwicklungsprozess (OOSE) geschaffen [10]. Hierunter wird verstanden, dass möglichst allgemein programmierte Objekte ihre Eigenschaften über eine Vererbung an die der konkreten Aufgabe angepassten Objekte weitergeben. Allgemeine Funktionalitäten können so in generalisierten Objekten untergebracht werden und wenn diese später für eine andere Softwareentwicklung benötigt werden, müssen diese nicht mehr neu programmiert werden und können so einfach von einem bereits existenten Objekt vererbt werden.

Es existieren mittlerweile umfangreiche Sammlungen von vorab konfektionierten Objekten, so genannte Klassenbibliotheken, von denen sicherlich die prominenteste die Microsoft Foundation Class ist, eine Klassenbibliothek, welche die Entwicklung von Programmen unter Microsoft Windows unterstützt. Viele Programmierer nutzen diese objektorientierten Ansätze nur als Erleichterung ihrer bisherigen Programmiertechnik und machen sich keine Gedanken darüber, wie ihre Problemstellungen generalisiert werden können um Eingang in eine Klassenbibliothek zu finden, sondern erstellen nur sehr spezialisierte Objekte. Auch sind die entstehenden Programme noch immer monolithisch, auch wenn Teile des Codes über so genannte Dll's[6] ausgelagert werden. Das eigentliche Ziel des Software-Engineering wird damit noch nicht erreicht, nämlich Software ebenso wie materielle Produkte gewissermaßen aus Normteilen zusammenzusetzen.

[6] Dynamic Link Library; dynamisch verknüpfbare Bibliothek

Diese Forderung wird durch den enormen Wandel der Softwareentwicklung in den letzten Jahren erheblich unterstrichen. Der Wandel wurde ausgelöst zum einen durch die sich stetig verbessernden Werkzeugen für die Entwicklung und zum anderen durch die ständig wachsenden Forderungen der Anwender hinsichtlich des Benutzerkomfort. Letzteres wurde unter anderem auch durch die enorme Zunahme von EDV-Wissen beim Anwender ausgelöst. Der heutige Prozess ist weder linear noch einfach iterativ sondern vielmehr sind wechselseitige Beziehungen in allen Teilbereichen des Softwarelebenszyklus (vgl. Abb. 3.1) an der Tagesordnung.

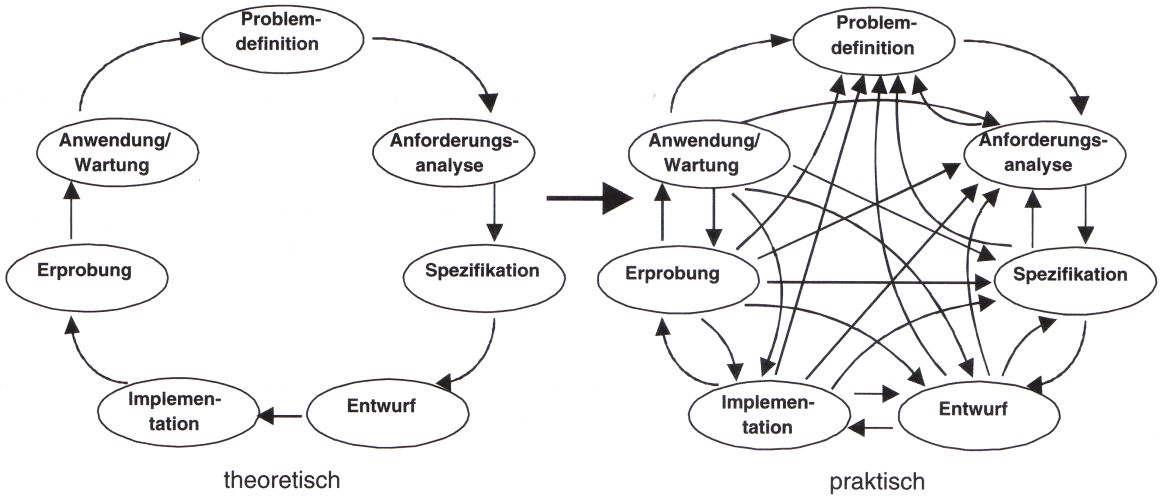


Abbildung 3.1: Phasenverlauf beim Softwarelebenszyklus [2]

Dies gilt es in einem neuen ganzheitlichen Ansatz zu realisieren. Hierbei kommt sowohl den erzeugten Daten (Dokumente, Listen, Tabellen, tec.) als auch den einzelnen Prozessen, welche diese Daten erzeugen, eine gleich große Bedeutung zu. Deshalb sollen zunächst die Anforderungen an das Daten- und Prozessmanagement formuliert werden.

3.1 Umfang des SLM

Das Software Lifecycle-Management-System (SLM) muss den gesamten Lebenszyklus eines Software Produktes umfassen. Dies beinhaltet sowohl die Unterstützung der Entwicklungs-

phase als auch der „*Produktivphase*“. Während diesen Phasen erfolgt eine ständige Weiterentwicklung der Software, sowie der zugehörigen Dokumente, wie Handbücher und ähnliches. Das SLM stellt kein Werkzeug zur Erstellung von ausführbaren Computerprogrammen dar, wie beispielsweise eine Entwicklungsumgebung für eine spezielle Programmiersprache, sondern ein übergeordnetes Hilfsmittel zur Lösung der Probleme, die bei der Erstellung und Verwaltung der heute immer komplexer werdenden Software auftreten.

Im Folgenden werden die Anforderungen an das SLM-System aufgrund der bisher aufgeführten Defizite im Bereich der Softwareentwicklung und –verwaltung formuliert.

3.2 Anforderungen an das Datenmanagement

Einen zentraler Bereich des SLM stellt das Datenmanagement dar. Das Datenmanagement hat die Aufgabe eine sinnvolle Verwaltung aller zu berücksichtigenden Dokumente zu gewährleisten. Im Bereich des Datenmanagements ist zunächst festzustellen, dass die Art der zu verwaltenden Dokumente je nach konkretem Anwendungsfall unterschiedlich sein kann. Das SLM muss daher konfigurierbar sein, um beliebige Dokumentarten zu verwalten. Die reine Verwaltung aller relevanten Dokumente ist bereits eine anspruchsvolle Aufgabe, ein Großteil der Informationen wird jedoch gerade dadurch gewonnen, dass Zuordnungen zwischen den einzelnen Dokumenten bestehen. Das SLM muss also in der Lage sein, beliebige Verbindungen oder auch Links, zwischen verschiedenen Dokumenten zu verwalten. Dadurch können Dokumente leicht gefunden werden, in dem der Anwender einem Link von einem anderen Dokument ausgehend folgt.

3.2.1 Verwaltung des Quellcodes

An erster Stelle der zu berücksichtigenden Dokumente stehen diejenigen Dokumente, welche den Quelltext der Software enthalten. Da diese die Algorithmen eines Computerprogramms beinhalten, stellen sie den Kern eines Software-Produktes dar. Die Quellcodes bilden den Ausgangspunkt für die Erstellung eines lauffähigen Gesamtsystems. Bei der Mehrzahl der heute verwendeten Programmiersprachen wird jedoch das ausführbare Computerprogramm

durch komplexe Übersetzungsvorgänge aus den Quelltextdateien erstellt. Die Zugehörigkeit des ausführbaren Programms zu den Quelltexten, die für seine Erstellung verwendet wurden, ist nur eine der zentralen Anforderungen, die sich für das Datenmanagement eines SLM-Systems ergeben. Für eine sinnvolle Verwaltung im Sinne des SLM lassen sich die folgenden konkreten Anforderungen aufstellen:

In den vorhergehenden Abschnitten wurde bereits dargelegt, dass die Softwareentwicklung keinesfalls als ein sequentieller Prozess gesehen werden kann, der mit der Erstellung und Auslieferung eines Computerprogramms beendet ist. Vielmehr ist der Entwicklungsprozess als eine kontinuierliche Weiterentwicklung eines Computerprogramms zu sehen. Dabei ist es zunächst unerheblich, ob es sich um funktionale Weiterentwicklungen oder um Fehlerbereinigungen handelt. Wesentlich ist in diesem Zusammenhang, dass im Laufe der Zeit verschiedene Revisionen des Produktes entstehen. Diese Revisionen umfassen alle zu berücksichtigenden Dokumente und Informationen; sowohl die eigentlichen Programme und Quelltextdateien, als auch alle anderen zugehörigen Dokumente wie Benutzerhandbücher, Installationsanleitungen etc.. Auch vor dem Aspekt, dass ein Unternehmen nicht davon ausgehen kann, dass alle Kunden ständig mit der neuesten Revision eines Software-Produktes arbeiten, ist die Verfügbarkeit älterer Revisionen für den Support unerlässlich. In engem Zusammenhang mit der Verwaltung von Revisionen eines Software-Produktes stehen auch Aspekte der Qualitätssicherung. So muss beispielsweise definiert sein, wann eine Revision eines Produktes freigegeben werden kann.

Der Ablauf bei der Erstellung eines auslieferungsfähigen Software-Produktes besteht vereinfacht aus dem Übersetzen der Quelltextdateien zu einem lauffähigen Computersystem. Im einfachsten Fall kann eine daraus resultierende ausführbare Datei (das Programm oder eine EXE-Datei) auf einen beliebigen Computer kopiert und dann dort verwendet werden. In der heutigen Praxis ist der Installationsvorgang jedoch um ein Vielfaches komplexer. Das eigentliche Computerprogramm besteht in der Regel nicht nur aus einer ausführbaren Datei, sondern aus einer Vielzahl von weiteren Dateien. Dabei kann es sich sowohl um eigene Module in Form von Bibliotheken handeln, als auch um bestimmte Standardbibliotheken, die auf dem Zielrechner vorhanden sein müssen.

Ebenfalls zum Lieferumfang gehört die Dokumentation, die ebenfalls in Form von Dateien mit installiert werden muss. Abhängig von der Art des Software-Produktes kann es auch erforderlich sein, bestimmte Dateien in ganz spezielle Verzeichnisse auf dem Zielrechner zu kopieren. Neben dem Kopieren von Dateien, müssen im Rahmen des Installationsvorgangs oft auch Einstellungen des Betriebssystems für das Software-Produkt verändert oder neu hinzugefügt werden. Dazu kann die Eintragung von Systemvariablen gehören, oder im Falle des Betriebssystems Windows, Einträge in der zentralen Registrierungsdatenbank des Systems. Diese Aspekte machen deutlich, dass der Installationsvorgang durchaus ein komplexer Prozess sein kann, der für das einwandfreie Funktionieren des Software-Produktes eine Vielzahl von kritischen Stellen beinhaltet.

Um den Vorgang der Installation möglichst sicher ausführen zu können, ist ein automatisches Installationsprogramm erforderlich, welches dem Benutzer den Grossteil dieser Aufgaben abnimmt, und damit die Zahl der möglichen Fehlerquellen reduziert. Dieses Installationsprogramm ist für sich selbst auch wieder ein Computerprogramm, welches dazu dient, dass Software-Produkt in einem funktionsfähigen Zustand auf ein Rechnersystem zu übertragen. Die Erstellung eines solchen Installationsprogramms kann mit der Fertigung eines „klassischen“ Produktes verglichen werden. Das SLM muss in der Lage sein, Installationsprogramme zu verwalten und zu einem beliebigen Zeitpunkt die Erstellung eines Installationsmediums zu initiieren. Die erstellten Installationsprogramme müssen natürlich der ihr zugrunde liegenden Revision des Software-Produktes zugeordnet sein und damit ebenfalls der Revisionsverwaltung unterliegen.

Da es sich bei einem Software-Produkt um ein System handelt, welches einer ständigen Weiterentwicklung unterliegt, ist es im Sinne der Qualitätssicherung relevant, dass die Änderungen, welche an den Quelltexten oder an anderen Dokumenten vorgenommen wurden, jederzeit nachvollziehbar sind. Das SLM muss deshalb in der Lage sein, die Änderungen hervorzuheben und darzustellen, die beispielsweise von einer Revision einer Quelltextdatei zur nächsten vorgenommen wurden.

3.2.2 Realisierung eines Konfigurationsmanagements

Das zu erstellende Computerprogramm muss unter Umständen für unterschiedliche Zielplattformen entwickelt werden. Beispielsweise ist es nicht ungewöhnlich, dass ein Programm sowohl unter dem Betriebssystem Microsoft-Windows als auch unter einem UNIX-Betriebssystem lauffähig sein muss [11]. In der Regel erfordert eine andere Zielplattform mindestens auch andere Übersetzungsroutinen für die Erstellung eines Programms aus den Quelltexten (Wenn die Zielplattformen „*Source-Code-kompatibel*“ sind). Die Zielplattform kann bereits eine andere Art der Codierung erfordern. Beispielsweise ist für die Programmierung einer Benutzeroberfläche für das Betriebssystem Windows eine ganz unterschiedliche Programmierung erforderlich als für UNIX. Mit Blick auf die Quellcode-Dateien bedeutet dies, dass auch hier unterschiedliche Versionen vorhanden sein müssen. Das SLM muss die Möglichkeit bieten, unterschiedliche Versionen desselben Programms zu verwalten, die teilweise aus den gleichen, teilweise jedoch auch aus unterschiedlichen Quelltextdateien bestehen.

Damit existieren dann unterschiedliche Konfigurationen des selben Software-Produktes. Unterschiedliche Konfigurationen können jedoch auch aus einer modulartigen Struktur des Software-Produktes resultieren. Beispielsweise kann die Verwendung einer bestimmten Quelltextdatei A die Verwendung einer Bibliothek X für eine erfolgreiche Übersetzung erfordern, wogegen bei Verwendung einer Quelltextdatei B die Bibliothek Y eingebunden werden muss.

3.2.3 Support- und Dokumentationsmöglichkeiten erstellter Software

Wie bereits in Kapitel 2.4 dargestellt, kann die Bedeutung der Dokumentation nicht überschätzt werden. Dabei ist zu berücksichtigen, dass es unterschiedliche Arten von Dokumentation zu einem Software-Produkt gibt. Neben der für den Anwender wichtigen Dokumentation in Form von Installations- und Bedienungsanleitungen, spielt vor allem die für die Entwickler gedachte Programmdokumentation im Hinblick auf die Wartungs-freundlichkeit und Erweiterbarkeit des Software-Produktes eine zentrale Rolle. Genau wie die

bisher erwähnten Dokumente müssen alle Dokumentationsinformationen revisionsbezogen verwaltet werden. Darüber hinaus ist der folgende Aspekt für eine effiziente Weiterbearbeitung eines Software-Produktes von entscheidender Bedeutung.

Besonders die Programmdokumentation ist für den Entwickler wichtig, wenn er vor dem Problem steht, eine bestimmte neue Funktionalität in ein bestehendes System zu integrieren oder die Funktionsweise eines Programms zu verstehen, wenn dieses Programm von einem anderen Entwickler erstellt wurde. Die heute übliche Form der Programmdokumentation, welche aus Inline-Kommentaren oder einer externen Beschreibung oder aus einer Mischform von beidem besteht, ist nicht dazu geeignet die Transparenz der Quelltexte zu erhöhen. Wünschenswert ist eine Möglichkeit, von einem bestimmten Teil des Quelltextes die entsprechende Dokumentation zu erreichen und umgekehrt. Dazu ist eine Zuordnung der Programmdokumentation zu den Bereichen der Quelltexte erforderlich, die nicht nur auf Dateiebene stattfindet, sondern die einen Verweis auf exakt bezeichnete Bereiche sowohl des Quelltextes als auch der Programmdokumentation gestattet.

Da sich gezeigt hat, dass die Handhabung der unterschiedlichen Arten der Dokumentation gerade in den nachgelagerten Phasen des Lebenszyklus eines Software-Produktes von ganz entscheidender Bedeutung sind, werden im folgenden explizit weitere Anforderungen formuliert, welche an das SLM gestellt werden.

Um die Erstellung der jeweils nötigen Dokumentation zu unterstützen, soll das SLM für jede neu erstellte Komponente des Software-Produktes zumindest eine Dokumentationsvorlage in Form eines Formblattes erstellen. Auf diese Art kann dem Effekt entgegengewirkt werden, der bei vielen Entwicklern zu beobachten ist, dass die Erstellung der Dokumentation auf den Zeitpunkt verschoben wird, wenn das Projekt abgeschlossen ist. Zu diesem Zeitpunkt ist es nicht ungewöhnlich, dass aus verschiedenen Gründen andere Tätigkeiten eine höhere Priorität erlangen und somit die betreffende Dokumentation gar nicht erstellt wird.

Das SLM muss außerdem in der Lage sein, Dokumentationen in beliebiger Form zu integrieren, wie beispielsweise Flussdiagramme, Konzepte, Ablaufbeschreibungen oder die Ansichten von Bildschirmfenstern.

Es wurde bereits an früherer Stelle erwähnt, dass insbesondere die Erstellung der Dokumentation oftmals ein kritischer Faktor ist, weil ihre Erstellung und ihre Qualität von der Disziplin der beteiligten Personen abhängen. Das SLM muss, soweit möglich, hier unterstützend eingreifen. Dies geschieht, wie im vorhergehenden Abschnitt beschrieben, durch die automatische Erstellung von Dokumentationsformblättern. Damit ist jedoch das Spektrum noch nicht ausgeschöpft. Die Unterstützung des SLM durch eine automatische Generierung von Vorlagen darf nicht nur auf Dokumentationsformblätter beschränkt sein.

Sinnvoll ist auch die automatische Erstellung anderer weitergehender Dokumente, wie beispielsweise eines Pflichtenheftes, welches automatisch als Vorlage erstellt wird, wenn ein neues Projekt begonnen wird. Dadurch kann außerdem gewährleistet werden, dass alle Pflichtenhefte nach demselben Schema aufgebaut sind und die notwendigen Kapitel enthalten, so dass die unbedingt notwendige Struktur der Informationen gegeben ist. Ein anderes Beispiel ist die automatische Erstellung von Vorlagen für weitere Dokumentationsarten, die automatisch vom SLM vorgenommen wird, wenn neue Quelltextdateien erzeugt werden.

Die zentrale Aufgabe des Supports ist die Unterstützung des Kunden bei der Anwendung des Software-Produktes. Dies reicht von einer Installation der Software über die Betreuung des Kunden im Hinblick auf die Bedienung. Dazu gehört in der Regel eine Unterstützung in dem Falle, dass der Kunde ein Problem an den Support meldet, welches dort bearbeitet und möglichst umgehend gelöst wird. Bei diesen Problemen kann es sich einerseits um Bedienungsfehler handeln, um ein fehlerhaftes Verhalten des Systems aufgrund einer falschen Konfiguration oder um einen Fehler innerhalb des Software-Produktes. Der Support muss also zunächst eine Klassifizierung des Problems durchführen. Je nach der Komplexität des Software-Produktes und der Art des gemeldeten Fehlers ist jedoch bereits die Reproduktion eines Fehlers auf einem System des Anbieters oftmals ein Problem. Die Situation, dass ein bestimmter Fehler nur in der Installation eines Kunden auftritt, nicht aber innerhalb des Testsystems des Produktsupports erschwert eine Lösung des Problems erheblich. Gerade bei Software-Produkten, die aus mehreren komplexen Komponenten mit einer Vielzahl von Konfigurationseinstellungen bestehen, ist es oftmals unerlässlich die identische Umgebung

einer spezifischen Kundeninstallation herzustellen, um einen Fehler zu lokalisieren und in der Folge auch zu beheben.

Das SLM muss aus diesem Grunde in der Lage sein, beliebige Kundenumgebungen zu verwalten und diese bei Bedarf auch zu aktivieren. Dabei umfasst der Begriff Kundenumgebung hier alle relevanten Komponenten, also beispielsweise die beim Kunden eingesetzte Revision des Software-Produktes sowie je nach Art des Produktes, die entsprechende Version des Betriebssystems, eventuell verwendete Datenbanksysteme, inklusive authentischer Datenbestände und alle anderen Computersysteme, die mit dem Software-Produkt in Verbindung stehen. Hier wird sofort deutlich, dass die Forderung nach der Verwaltung von Kundenumgebungen für Supportzwecke eine große Komplexität beinhaltet und einen beträchtlichen Aufwand darstellt. Gerade bei komplexen Systemen mit einem gewissen Grad an kundenspezifischer Individualität ist dies jedoch oft absolut notwendig, um in einer für den Kunden akzeptablen Zeit eine Lösung zu finden.

Neben der Verwaltung von Kundenumgebungen, die nicht für jede Art eines Software-Produktes erforderlich ist, müssen jedoch in jedem Fall weitere Informationen über die Kunden vorhanden sein. Dazu gehört natürlich die Information darüber, welche Revision des Produktes bzw. welche Module des Produktes in Betrieb sind. Daneben ist je nach Vertriebsmodell des Software-Produktes auch festzuhalten, welche Anzahl Lizenzen für die Software der Kunde erworben hat oder ob ein Wartungsvertrag abgeschlossen wurde, der zur Inanspruchnahme des Supports erforderlich ist.

Da für den Support eines Software-Produktes in der Regel mehrere Mitarbeiter zuständig sind, ist auch die einheitliche Erfassung aller bearbeiteten Supportanfragen zwingend notwendig, damit beispielsweise nicht unnötiger Aufwand getrieben wird, um ein Problem zu lösen, welches bereits in der Vergangenheit von einem anderen Mitarbeiter gelöst wurde.

Für die Lokalisierung und Behebung von Fehlern ist es oft auch notwendig, dass der Support Einblick in die Quelltexte hat. Dabei ist zunächst unerheblich, ob der Support die Berechtigung hat, die Quelltexte zu ändern, oder ob dies nur von den Mitarbeitern der Entwicklungsabteilung durchgeführt werden kann. Gerade im Support ist die bereits oben

angesprochene Verknüpfung der Dokumentation zu den Quelltexten von enormer Bedeutung. Die Möglichkeit eine Fehlerquelle, ausgehend von der Dokumentation und einer Fehlerbeschreibung direkt in den Quelltexten zu finden, ist ungleich schneller als mit konventionellen Mitteln ohne spezielle Unterstützung durch das SLM. Die im SLM erfassten Fehler können genauso wie die Dokumentation über Verweise mit den entsprechenden Stellen im Quelltext verknüpft werden.

3.3 Anforderungen an das Prozessmanagement

Das SLM muss generell den gesamten Lebenszyklus eines Software-Produktes abdecken. Dies beinhaltet die Phasen der Konzeption, der eigentlichen Programmierung, aber auch die nachgelagerten Bereiche, wie die ständige Weiterentwicklung und den Support des Software-Produktes, welches bei den Kunden installiert und in Betrieb ist.

Der Übergang eines Software-Produktes von dem Stadium der Entwicklung in eine Testphase und die anschließende Erstellung von Installationsmedien, setzt voraus, dass Prozesse analog der konventionellen Produktion initiiert und gesteuert werden. Im Umfeld der klassischen produzierenden Unternehmen haben sich für die Steuerung und Überwachung von Geschäftsprozessen so genannte Workflow-Systeme etabliert. Besonders weil der Softwarelebenszyklus nicht aus sequentiell ablaufenden Prozessen besteht, sondern, wie zu Beginn dieses Kapitels dargestellt, in der Praxis aus einem Netzwerk sich teilweise überlappender Phasen, ist es absolut notwendig, dass die Steuerung dieser Prozesse durch das SLM gewährleistet wird.

Das SLM muss also über ein Workflow-Modul verfügen, welches in der Lage ist, frei definierbare Prozesse zu steuern und zu überwachen. Dabei wird unter einem Prozess in diesem Zusammenhang die folgende Funktionalität (vgl. Abbildung 3.2) verstanden:

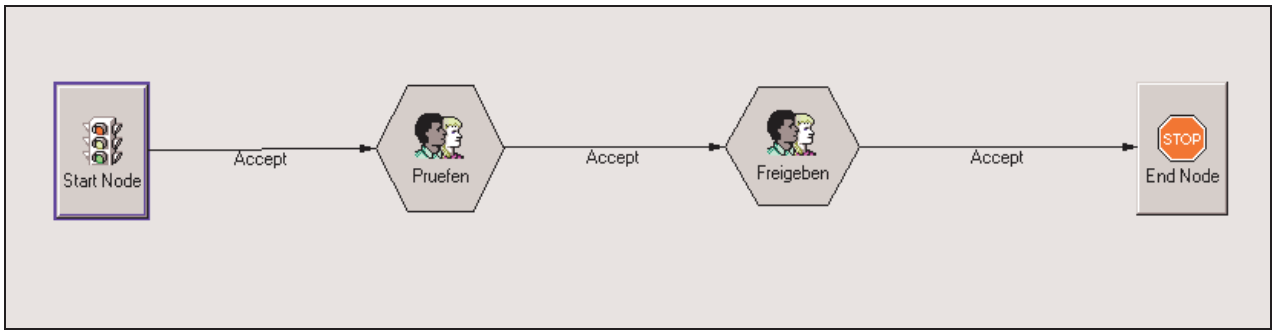


Abbildung 3.2: Prozess-Definition

Beliebige Dokumente können über Links mit einem Prozess verbunden werden. Auf diese Art können dem Prozess Informationen hinzugefügt werden, die innerhalb des Unternehmens zwischen verschiedenen Mitarbeitern ausgetauscht werden müssen. Ein Prozess ist somit vergleichbar mit einer elektronischen Umlaufmappe.

Ein Prozess besitzt einen definierten Anfangs- und Endknoten. Dazwischen können beliebige andere Knoten bzw. Stationen angeordnet werden. Jedem Knoten kann ein Benutzer oder eine Benutzergruppe zugeordnet sein. Ein Start des Prozesses bewirkt, dass der Prozess zu dem ersten definierten Knoten läuft, wodurch die entsprechenden Benutzer an diesem Knoten eine Mitteilung darüber erhalten, dass ein neuer Prozess eingetroffen ist. Übernimmt ein Mitglied der Benutzergruppe den Prozess, so übernimmt er automatisch in diesem Moment die Kontrolle über den Prozess. Neben den Benutzern kann jedem Knoten eine oder mehrere Aufgaben zugewiesen werden. Nach Erledigung der Aufgaben kann der Prozess an die nächste Station weitergeleitet werden. Dabei kann der Bearbeiter dem Prozess auch weitere Dokumente hinzufügen. Sollte eine Aufgabe nicht ausgeführt werden können, besteht für den jeweiligen Bearbeiter immer auch die Möglichkeit, den Prozess nicht weiterzuleiten, sondern ihn zurückzuschicken.

Ein Beispiel für den Einsatz eines solchen Workflow-Prozesses ist der Vorgang der Erstellung von Installationsmedien. Nachdem der Entwickler die Codierung und die ersten Tests einer Funktionalität abgeschlossen hat, muss eine Installationsversion des Computerprogramms erstellt werden, welche anschließend für die Tests der Software verwendet wird. Der Entwickler erstellt also einen neuen Setup-Prozess und fügt diesem die Quelltexte sowie die anderen erstellten Dokumente hinzu. Nachdem dies geschehen ist, startet er den Prozess. Der

Setup-Prozess wird per Definition an die Benutzer weitergeleitet, welche in der „Produktion“ beschäftigt sind. Der zuständige Mitarbeiter öffnet den Prozess und findet alle notwendigen Dokumente, die er zur Erledigung der Aufgabe benötigt, direkt in der Prozessansicht.

Bereits dieses einfache Beispiel macht deutlich, welchen Nutzen der Einsatz eines Workflows im Hinblick auf die Kommunikation und Qualitätssicherung hat. Der Aufwand für die Suche nach den richtigen Revisionen von Dokumenten kann drastisch gesenkt werden, ebenso wie die Anzahl der möglichen Fehlerquellen. Da die Anzahl und die Art der abzubildenden Prozesse stark von dem jeweiligen Unternehmen abhängen, ist es erforderlich, dass Prozesse innerhalb des SLM frei definiert werden können.

Die Realisierung eines Workflows setzt eine Netzwerkfähigkeit des SLM voraus, da die unterschiedlichen Mitarbeiter natürlich in der Regel auch an unterschiedlichen Rechnern innerhalb des Unternehmensnetzwerks arbeiten. Ebenso ist für den Einsatz des Workflows ein Benachrichtigungssystem notwendig, welches die Benutzer darüber informiert, dass ein neuer Prozess eingetroffen ist. Dieses Benachrichtigungssystem kann in das SLM integriert werden. Alternativ ist auch die Verwendung eines konventionellen Standard-Email-Systems für diese Aufgabe möglich.

Das Workflow-Modul des SLM muss auch mit einer WEB-basierten Benutzeroberfläche versehen werden. Dies bietet den Vorteil, dass keinerlei Installation auf den einzelnen Rechnern erforderlich ist. Der größere Nutzen einer Web-Oberfläche tritt jedoch vor allem dann in Erscheinung, wenn Unternehmen im Bereich der Softwareentwicklung oft mit externen Unternehmen oder freien Mitarbeitern zusammenarbeiten. Wenn der Workflow mit Hilfe der Web-Technologie umgesetzt wird, ist die Integration von externen Mitarbeitern in die Geschäftsprozesse leicht umzusetzen.

Daneben muss das Workflow-Modul des SLM die Möglichkeit bieten, eine Übersicht über die Prozesse zu erstellen. Im Hinblick auf das Projektmanagement kann der Projektleiter somit erkennen, wie der Fortschritt eines Projektes ist. Durch diese Erhöhung der Transparenz der Projektabwicklung kann das SLM auch wertvolle Dienste als Tool für eine Optimierung der Geschäftsprozesse leisten. Eine Verbesserung kann nur erreicht werden, wenn

Information darüber vorliegen, wann und wo in der Vergangenheit ein Optimierungspotenzial erkennbar war. Diese Analyse kann mit Hilfe des SLM einfach dadurch erfolgen, dass die abgeschlossenen Prozesse über einen bestimmten Zeitraum im Nachhinein untersucht werden.

3.4 Unterstützung der Kommunikation von verschiedenen Entwicklern

Die vorhergehenden Kapitel haben bereits gezeigt, dass gerade die Koordination verschiedener Mitarbeiter, die gleichzeitig an einem Projekt arbeiten, viele Probleme aufwerfen können. Daher sollen in diesem Kapitel explizit die Anforderungen an das SLM beschrieben werden, die zur Umsetzung eines „*Concurrent Development*“ erforderlich sind. Viele der bisher formulierten Anforderungen unterstützen bereits die Koordination von mehreren Mitarbeitern während der Entwicklung eines Software-Produktes, wie zum Beispiel das beschriebene Workflow-System.

Grundlage für die Koordination der einzelnen Mitarbeiter ist zunächst eine Benutzerverwaltung, die den Zugang der Mitarbeiter zum SLM steuert. Die im System bekannten Benutzer müssen in Gruppen zusammengefasst werden können. Eng mit der Benutzerverwaltung verknüpft ist eine Zugriffsrechteverwaltung, Diese Zugriffsrechte-verwaltung ist bei allen Aktionen involviert, welche die im SLM abgelegten Dokumente und Informationen betreffen. Dabei muss vor jeder Durchführung einer Aktion festgestellt werden, ob der betreffende Benutzer das Recht hat, diese Aktion mit einem bestimmten Objekt durchzuführen. Für diese Feststellung wird auch geprüft, ob der Benutzer auf Grund seiner Zugehörigkeit zu einer Gruppe ein Zugriffsrecht erhält.

Die Vergabe der Zugriffsrechte erfolgt zunächst auf der Basis der Klassenzugehörigkeit. So muss beispielsweise festgelegt werden, dass Mitglieder der Gruppe „Support“ Quelltext-Dokumente zwar ansehen, nicht jedoch verändern dürfen.

Für eine effiziente Rechteverwaltung reicht jedoch diese Form der Zugriffssteuerung nicht aus. In der Praxis hängt die Frage, ob eine bestimmte Aktion mit einem speziellen Objekt ausgeführt werden darf, auch von dem momentanen Zustand ab, in dem sich das Objekt befindet. Es ist also darüber hinaus notwendig für jedes verwaltete Objekt des SLM einen

Status mitzuführen. In diesem Zusammenhang wird auch von einem sogenannten „Lifecycle-State“ gesprochen. Dieser Lebenszyklus-Zustand kann beispielsweise die folgenden Werte annehmen:

Neu, Eingecheckt, In Bearbeitung, Freigegeben, Ungültig

Bereits an Hand solcher definierter Zustände lassen sich einfache Regeln formulieren, wie beispielsweise:

„Eine freigegebene Quelltextdatei darf nicht mehr geändert werden!“

Der positive Effekt eines solchen Mechanismus wird anhand des folgenden Szenario sofort deutlich:

> *Ein Entwickler hat die Arbeit an den Quelltexten beendet und die Mitarbeiter des Supports haben bereits die Setup-Version des Software-Produktes mit den ihnen vom Entwickler übergebenen Quelltexten abgeschlossen. Nun fällt dem Entwickler auf, dass in einer der Quelltextdateien noch ein Fehler enthalten ist, den er umgehend korrigiert. Da er jedoch die Mitarbeiter der Support-Abteilung nicht darüber informiert, liegt nun ein inkonsistenter Zustand vor. Die Auswirkungen einer solchen Situation können unter Umständen katastrophal sein, wenn auf diese Art eine fehlerhafte Version des Software-Produktes in den Vertrieb gelangt. Zumindest sind jedoch Verwirrung und ein unnötiger Aufwand zur Bereinigung dieser Situation die Folge.*

Bei Einsatz einer Zugriffsregelung im Zusammenwirken mit einer Lebenszyklusverwaltung kann die oben beschriebene Situation nicht mehr eintreten, da hierbei beispielsweise die folgenden Regeln eingehalten werden müssen:

- **Eine Setup-Version kann nur erstellt werden, wenn alle zugehörigen Quelltextdateien im Zustand „Freigegeben“ sind.**

- **Ein Dokument, welches im Zustand „Freigegeben“ ist, kann nicht bearbeitet werden.**

Die Verwendung eines Lebenszyklus verhindert auch, dass Informationen dadurch verloren gehen, dass zwei Entwickler gleichzeitig dieselbe Datei bearbeiten. Ohne eine Unterstützung durch das SLM würde einer der Entwickler zwangsläufig die Änderungen seines Kollegen, der die Datei zuvor gespeichert hat, überschreiben, wobei dies keinem der Beteiligten auffallen würde. Innerhalb des SLM muss der Entwickler zunächst eine Datei auschecken, bevor er diese bearbeiten kann. Durch das Auschecken der Datei bekommt diese automatisch den Zustand „In Bearbeitung“, wodurch sie für andere Benutzer gesperrt ist. Je nach Konfiguration kann das System zwar noch einen lesenden Zugriff gestatten, ein Überschreiben ist jedoch in jedem Falle ausgeschlossen.

Das hier beschriebene Freigabeschema mit den zu Grunde liegenden Stati und Berechtigungen muss darüber hinaus von dem SLM konfigurierbar gehalten werden, damit individuelle Regeln des Unternehmens auch abgebildet werden können.

3.5 Oberfläche des SLM

Die Benutzeroberfläche stellt die Schnittstelle zwischen dem SLM und den Anwendern dar. Von ihrer Leistungsfähigkeit hängt zum großen Teil die Akzeptanz des gesamten Systems durch die Anwender ab. Die Benutzeroberfläche muss den heutigen Standards von grafischen Benutzeroberflächen entsprechen, wie sie bei den aktuellen Betriebssystemen, wie zum Beispiel Windows, Stand der Technik sind. Da es sich hierbei um Eigenschaften handelt, die jedem Anwender eines PC bekannt sind, soll auf die Anforderungen bezüglich der Standardfunktionalitäten an dieser Stelle nicht näher eingegangen werden. Im Zusammenhang mit dem SLM ergeben sich jedoch einige spezielle Anforderungen, die im folgenden beschrieben werden.

Die Möglichkeit zum Editieren von Quelltexten muss innerhalb des SLM gegeben sein. Zum Bearbeiten der Quelltextdateien ist kein externes Entwicklungsprogramm notwendig. Die Entwicklungsumgebung des SLM verfügt über die Möglichkeit die Quelltexte in intelligenter

Form anzuzeigen. Dies bedeutet, dass bei der Anzeige bereits eine erste Syntaxprüfung stattfindet. Schlüsselworte der Programmiersprache werden dann farblich hervorgehoben. Dies erhöht die Lesbarkeit der Quelltexte und stellt eine Arbeitserleichterung für den Entwickler dar. Das SLM muss weiterhin die Möglichkeit bieten, mehrere Quelltextdateien gleichzeitig zu öffnen.

Diese Anforderungen werden auch von den meisten Editoren der modernen Standard-Entwicklungsumgebungen erfüllt. An das SLM werden jedoch auch weitergehende Anforderungen gestellt, welche nicht mit Standardwerkzeugen abgedeckt werden können.

Die Benutzeroberfläche muss ebenfalls über die notwendigen Mechanismen verfügen, um aus den Quelltexten ein ausführbares Programm zu erstellen. Dies ist für die Tests der Software unbedingt erforderlich. Die Realisierung dieses Mechanismus muss frei konfigurierbar gestaltet sein, damit alle denkbaren Konfigurationen eines Softwaresystems abgedeckt werden können. Die bereits erwähnten Lifecycle-Operationen, wie das Ein- oder Aus-Checken von Quelltextdateien müssen aus der Entwicklungsumgebung aufrufbar sein.

3.6 Berücksichtigung unterschiedlicher Programmiersprachen

Generell muss das SML in der Lage sein, eine Software unabhängig von der verwendeten Programmiersprache zu verwalten. Um letztlich aus den Quelltexten ein ausführbares Gesamtsystem zu erstellen, müssen jedoch die zur Verfügung stehenden Übersetzungs-werkzeuge verwendet und somit aus dem SLM heraus aufgerufen werden können. Dies setzt voraus, dass eine Integration der in Frage kommenden Werkzeuge, wie beispielsweise eines C-Compilers vorhanden ist.

Die Forderung nach der Sprachunabhängigkeit des SLM ist von zentraler Bedeutung. Ein Großteil der Software-Produkte, besteht aus mehreren Teilsystemen, die oftmals in unterschiedlichen Programmiersprachen erstellt werden müssen. Als Beispiel seien Zusatzprogramme, auch sogenannte Plug-Ins, zu anderen Softwaresystemen genannt. Hier ist beispielsweise aus Gründen der Performance eine Programmierung in der Programmier-sprache C++ erforderlich. Andererseits ist die Anbindung an die Zielapplikation nur mit Hilfe

der Programmiersprache BASIC möglich. Beide Module sind eng mit einander verbunden. Da es sich bei der erstellten Software um ein Gesamtsystem handelt, ist auch die einheitliche Verwaltung zwingend erforderlich. Das SLM muss also auch innerhalb eines Projektes unterschiedliche Programmiersprachen und damit Entwicklungswerkzeuge unterstützen.

Zur Integration beliebiger Programmiersprachen und der damit verbundenen Programmierwerkzeuge, muss das SLM eine Schnittstelle bereitstellen, über welche die Einbindung erfolgen kann.

3.7 Zusammenfassung der Anforderungen

Die hier formulierten Anforderungen an ein SLM zeigen deutlich die Komplexität der zu lösenden Aufgabenstellung. Auf der einen Seite erzwingt der ständig steigende Wettbewerb eine Beschleunigung der Entwicklungszeit für neue Software-Produkte. Andererseits wird deutlich, dass ein großer Aufwand für die Erhaltung der Software erst nach der Fertigstellung eines Computerprogramms durch Fehlerbereinigung und Weiterentwicklung entsteht. Dies stellt nicht nur einen relevanten Aufwand dar, sondern kann im Extremfall dazu führen, dass ein Software-Produkt nicht mehr beherrschbar ist, da die Aufwendungen zur Lokalisierung und Beseitigung von Fehlern auf Grund der fehlenden Transparenz nicht mehr kalkulierbar sind. In solchen Fällen bleiben als Ausweg nur noch die Einstellung oder eine komplette Neuentwicklung des Produktes. Die Unterteilung der Anforderungen in mehrere Bereiche macht bereits deutlich, dass auch für das SLM ein modularer Aufbau gewählt werden muss. Die zu erwartende Komplexität des Softwaresystems SLM darf sich jedoch nicht nachteilig auf die Benutzeroberfläche auswirken. Da die Akzeptanz der Anwender generell ein wichtiges Kriterium für den erfolgreichen Einsatz eines Programms ist, muss das SLM über eine einfache intuitive Benutzeroberfläche verfügen. Bei der Konzeption der einzelnen Module muss außerdem berücksichtigt werden, dass auch eine schrittweise Einführung, beginnend mit einer Untermenge des Funktionsumfangs des SLM möglich ist, um eine schnelle Inbetriebnahme des Systems zu ermöglichen und einen signifikanten Produktivitätsknick zu vermeiden, den sich gerade kleine und mittlere Unternehmen in der Softwareentwicklung nicht leisten können.

Bevor jedoch das Konzept für das SLM aufgestellt werden kann, ist eine genaue Untersuchung der heute verfügbaren Technologien notwendig, um beurteilen zu können, in wie weit für die Realisierung des SLM auf bereits vorhandene Standards aufgebaut werden kann.

4 Aktuelle Technologien zur Verbesserung des Softwareentwicklungsprozesses

4.1 Allgemeines

Bei der Formulierung der Anforderungen ist bereits deutlich geworden, dass ein SLM selbst ein äußerst komplexes Programmsystem sein wird. Aus diesem Grund werden nun die heute zur Verfügung stehenden Technologien im Bereich der Software-Entwicklung untersucht, um eine geeignete Basis für das Konzept des SLM zu schaffen. Diese Untersuchung umfasst sowohl die Programmiersprachen, die für eine Umsetzung des Konzeptes geeignet sind, als auch die Technologien zur komponentenorientierten Softwareentwicklung.

Da ein Teil der in Kapitel 1 aufgestellten Anforderungen auch Teil der Funktionalität von kommerziellen Softwaresystemen ist, sollen diese Systeme ebenfalls dahingehend untersucht werden, ob sie als Teil des SLM eingesetzt werden können. Dies betrifft vor allem die Familie der PLM-Systeme, da hier eine ähnliche Aufgabenstellung gegeben ist.

4.2 Abgrenzung zu PLM-Systemen

PLM-Systeme sind hervorgegangen aus den EDM / PDM-Systemen [12]. Sowohl für EDM als auch für PDM existieren eine Vielzahl von Begriffen, die von *Electronic Document Management* bis hin zu *Product Data Management* reichen. Die Grenzen zwischen den einzelnen Definitionen sind fließend und variieren, je nach Anbieter und Leistungsfähigkeit des jeweiligen Systems. Heute wird für diese Familie der Softwaresysteme überwiegend der Begriff PLM oder *Product Lifecycle Management* verwendet, um die Bedeutung des gesamten Lebenszyklus zu betonen. Der theoretische Anspruch von PLM-Systemen lautet:

PLM-Systeme verwalten alle Daten, die mit einem Produkt in Zusammenhang stehen an zentraler Stelle während des gesamten Lebenszyklus des Produktes. Dies beinhaltet die Fähigkeit Arbeitsprozesse zu steuern und zu überwachen.

Nach dieser Definition könnte das SLM direkt in Form eines PLM-Systems erstellt werden, da ja bereits festgestellt wurde, dass auch die Entwicklung von Software eine Produktentwicklung darstellt. In der Tat existiert eine relativ große Überdeckung zwischen den Leistungen, die von PLM-Systemen erbracht werden und den Anforderungen, die an das SLM gestellt werden. In der Praxis ist jedoch festzustellen, dass der Schwerpunkt der PLM-Systeme in der Abbildung des klassischen Entwicklungsprozesses und hier im Besonderen in der Schnittstelle der CAD-Daten zu den nachfolgenden Fertigungsinformationen liegt. Dennoch ist eine genaue Auseinandersetzung mit den PLM-Systemen notwendig, um festzustellen, wie hoch der mögliche Überdeckungsgrad zu dem SLM ist.

Auch bei den PLM-Systemen findet sich ein modularer Aufbau, welcher auf der obersten Ebene zunächst das Datenmanagement auf der einen und das Prozessmanagement auf der anderen Seite enthält. Der Bereich des Datenmanagements ist in der Regel bei PLM-Systemen sehr umfassend. Es besteht die Möglichkeit Dokumente, wie CAD-Dateien, in einem geschützten Bereich zu verwalten und bei Bedarf für eine Bearbeitung auszuchecken. Um den Bedürfnissen moderner 3D-CAD-Systeme gerecht zu werden, können innerhalb eines PLM-Systems auch Beziehungen zwischen CAD-Dokumenten hergestellt werden, um in Baugruppen und Teilen auch Abhängigkeiten allgemein oder Abhängigkeiten von abgeleiteten 2D-Zeichnungen zu 3D-Modellen zu berücksichtigen. Daneben wird bei PLM-Systemen der Klassifizierung ein großer Stellenwert eingeräumt, da die sinnvolle Einteilung des Dokumenten- bzw. Teilespektrums eine unabdingbare Voraussetzung für die Wiederverwendung von abgelegten Informationen darstellt. Eine zentrale Aufgabe von PLM-Systemen ist auch die Unterstützung des Lebenszyklus, wie sie auch in ähnlicher Form für ein SLM gefordert wurde. Auch im Bereich des Workflows finden sich zahlreiche Überschneidungen mit den formulierten Anforderungen.

Bei allen Gemeinsamkeiten, die bei einem Vergleich der Anforderungen an ein SLM mit den Funktionen eines PLM-Systems festgestellt werden, treten jedoch auch einige schwerwiegende Defizite auf, die gegen eine Verwendung eines PLM-Systems im Rahmen der hier diskutierten Problemstellung sprechen.

PLM-Systeme sind traditionell auf den Bereich der Konstruktion und Fertigung im herkömmlichen Sinne fokussiert. Hier existieren ausgereifte Schnittstellen zu CAD-Systemen, um die erzeugten CAD-Daten zu erfassen und zu verwalten. Die für ein SLM notwendigen Schnittstellen müssten jedoch neu entwickelt werden. Da jedoch diese Schnittstellen tief in die Strukturen des PLM-Systems eingreifen müssen, um eine befriedigende Funktionalität bereitzustellen, sind umfangreiche Arbeiten und Manipulationen am PLM-System erforderlich. Die PLM-Systeme verfügen in der Regel nicht über ausreichend offene Programmierschnittstellen um solche tief greifenden Änderungen vornehmen zu können. Da es sich um hochspezialisierte Anwendungsprogramme handelt, bieten sie auch die Anpassungsmöglichkeiten in einer auf die eigentlichen Aufgabenstellung optimierten Form an. Die zur Verfügung stehenden Schnittstellen sind darauf ausgelegt, das sogenannte Customizing, also die Anpassung des PLM-Systems an eine spezifische Kundenumgebung, im herkömmlichen Kontext möglichst zu vereinfachen. Dies ist jedoch bei der Übertragung auf eine grundsätzlich andere Problemstellung wie das SLM nicht ausreichend, da hier andere Algorithmen implementiert werden müssen.

Ein anderer Bereich, der einer Verwendung eines PLM-Systems entgegensteht, ist die Abbildung des Problembereiches in Klassen des Systems. PLM-Systeme verfügen naturgemäß über eine hierarchische Klassenstruktur, welche Klassen wie 2D-Zeichnung, Baugruppe, Einzelteil, Normteil etc. enthält. Eine solche Struktur ist für das SLM ungeeignet und müsste komplett durch eine geeignete Klassenhierarchie ersetzt werden. Solche tiefgreifenden Eingriffe in die Basis der Datenstruktur eines Systems sind jedoch, wenn überhaupt möglich, nur mit einem hohen Aufwand umzusetzen. Ähnliches gilt für den Bereich des Workflow-Managements.

Neben diesen bedeutenden Gesichtspunkten, die eine Verwendung eines PLM-Systems betreffen, gibt es auch wirtschaftliche Faktoren, die nicht unberücksichtigt bleiben können. Die Kosten für ein PLM-System sind nicht unerheblich. Ein Kunde wäre gezwungen, diese Anschaffungskosten und laufende Lizenzgebühren für eine Software zu tragen, die zum einen nicht benötigte Funktionen enthält, wie beispielsweise Viewer-Programme für CAD-Dateiformate, deren Kernfunktionen andererseits weitgehend durch die SLM-spezifischen

Funktionen ersetzt sind. Eine solche Kombination lässt sich gerade bei kleinen und mittleren Unternehmen wahrscheinlich nicht einsetzen.

4.3 Grenzen von Datenbanksystemen

Aus den Anforderungen ergibt sich, dass ein zentraler Bestandteil des SLM die Speicherung und Verwaltung von großen Datenmengen ist. Heute haben sich im Bereich der technischen EDV die relationalen Datenbanken (RDB) etabliert [13]. Hier geschieht die Ablage der Daten in verschiedenen Tabellen, welche miteinander verknüpft werden können um Beziehungen zu verwalten. Auch wenn auf Grund der Art der zu verwaltenden Informationen objektorientierte Datenbanken besser geeignet wären, bieten die relationalen Datenbanken im Hinblick auf die Performance sowie die Flexibilität ausschlaggebende Vorteile. Auch die Forderung nach der Verwendung von Standards, spricht für die relationalen Datenbanksysteme, da sich im Bereich der objektorientierten Datenbanksysteme [14] bis Heute kein Standart etabliert hat.

Zusammengefasst bieten relationale Datenbanken die im Folgenden aufgeführten Eigenschaften:

Relationale Datenbanken sind in der Lage ein großes Datenvolumen mit einer guten Performance zu verwalten. Durch das Vorhandensein einer Client- / Server-Architektur ermöglichen sie auch weiterhin einen Zugriff über ein Netzwerk. Der Zugriff auf Informationen der Datenbank kann bei den meisten Systemen mit Hilfe einer Benutzer- und Gruppenverwaltung über grundlegende Zugriffsrechte gesteuert werden.

Ein wesentlicher Gesichtspunkt ist auch die Koordination, mehrerer gleichzeitiger Zugriffe auf abgelegte Informationen. Wie im Rahmen der Anforderungen beschrieben, ist es eine Kernfunktionalität des SLM, dass mehrere Mitarbeiter zeitgleich an demselben Projekt arbeiten können, was zwangsläufig bedeutet, dass ein Datensatz gleichzeitig von mehreren Benutzern verwendet werden kann. Relationale Datenbanken bieten hierfür den Mechanismus des sogenannten „*Record Locking*“. Das bedeutet, dass ein Datensatz von einem Benutzer reserviert werden kann. Sobald eine Reservierung erfolgt ist, kann dieser Datensatz von

anderen Benutzern nur noch gelesen, nicht aber verändert werden. Dies entspricht vom Prinzip her bereits dem Verhalten, welches auch in Kapitel 3.4 gefordert wurde.

Weiterhin steht mit SQL[7] eine standardisierte Programmiersprache für Zugriffe auf die Datenbank zur Verfügung, die von allen gängigen Datenbankmanagementsystemen unterstützt wird. SQL erlaubt die Erstellung einer Datenbank und die Abfrage und Manipulation von gespeicherten Daten [15]. Alle für das SLM relevanten Aktionen können prinzipiell mit Hilfe von SQL umgesetzt werden. Heute haben sich aufbauend auf SQL auch andere Zugriffstechniken etabliert, welche den Ansätzen einer objektorientierten und komponentenbasierten SW-Entwicklung entsprechen. Wenn auch auffällt, dass eine große Zahl der gestellten Anforderungen bereits prinzipiell mit Hilfe von relationalen Datenbanken abgebildet werden können, so ist doch festzustellen, dass alle Operationen im Umfeld der relationalen Datenbanken auf der Ebene einzelner Datensätze stattfinden.

Im Rahmen des SLM müssen jedoch Objekte verwaltet werden, die über vielfältige Beziehungen mit anderen Objekten in Verbindung stehen, wobei diese Beziehungen jeweils wiederum den Zustand von Objekten und die Aktionen auf Objekte beeinflussen. Eine Abbildung des SLM mit Hilfe eines relationalen Datenbanksystems würde eine äußerst komplexe, schwer zu beherrschende Anwendung hervorbringen. Andererseits stellt eine komplette Neuentwicklung auf der Basis der gestellten Anforderungen keine befriedigende Alternative dar, da die zu Grunde liegenden Funktionalitäten ja bereits in Form der relationalen Datenbanken vorliegen. Um mit vertretbarem Aufwand eine optimale Lösung zu erreichen, empfiehlt sich die Nutzung einer relationalen Datenbank durch das SLM, wobei das SLM über ein zentrales Modul verfügt, welches die Funktionen für den Datenbankzugriff bereitstellt. Dieses Modul kapselt die Datenverwaltung und bildet die Objektstruktur des SLM auf die Struktur der relationalen Datenbank ab.

[7] SQL: Structured Query Language

4.4 Möglichkeiten von CASE-Tools

Es existieren eine Vielzahl unterschiedlicher CASE[8]-Tools mit verschiedenen Funktionen [4]. Entstanden sind diese Tools überwiegend aus Überlegungen zu einer objektorientierten Softwareentwicklung, die zu Beginn der neunziger Jahre als Reaktion auf die sogenannte Softwarekrise begannen. Die größte Gruppe der CASE-Tools verfolgt die Zielrichtung, den Quelltext eines Computerprogramms im Hinblick auf die Kriterien der objektorientierten Theorie, wie Wiederverwendbarkeit oder Erweiterbarkeit zu optimieren. Desweiteren sind Funktionen vorhanden, die den Prozess der objektorientierten Analyse und des Designs unterstützen und eine konforme Erstellung des Quelltextes ermöglichen. Dies geht soweit, dass Diagramme und Ablaufpläne von der CASE-Anwendung in ein Gerüst in der jeweiligen Programmiersprache überführt werden, welches in der Folge von dem Entwickler weiter ausgebaut wird.

Da der Begriff CASE einen großen Interpretationsspielraum zulässt, existieren daneben auch eine Vielzahl andersartiger Programme, deren Funktionsweise von der oben beschriebenen abweicht. Die Zielrichtung weicht jedoch deutlich von den hier gestellten Anforderungen ab, da sie hauptsächlich auf die Architektur des Quelltextes ausgerichtet ist. So werden manchmal auch Software-Entwicklungsumgebungen, wie Microsoft Visual Studio als CASE Tools bezeichnet. Systeme, welche einen dem SLM ähnlichen Anspruch haben, finden sich in der Literatur unter der Bezeichnung „CASE-Umgebung“. Solche CASE-Umgebungen haben sich jedoch in der Praxis nicht durchsetzen können. In der Literatur finden sich folgende Merkmale für CASE-Umgebungen:

- Unterstützung mehrerer Phasen des Softwarelebenszyklus (einschließlich der Analysephase),
- Integration der einzelnen Werkzeugkomponenten über eine gemeinsame Benutzer- und Datenschnittstelle und
- Bereitstellung von grafischen Beschreibungsmitteln (ER-Diagramm, Datenflussdiagramm, Klassenhierarchie, etc.)

[8] CASE: Computer Aided Software Engineering

Der Hauptgrund für die schwache Verbreitung solcher CASE-Umgebungen ist der hohe Aufwand im Rahmen der Einführung, der durch Schulung und Einarbeitung der Mitarbeiter entsteht. Bei vielen EDV-Systemen kommt es im Zuge der Systemeinführung zunächst zu einem Produktivitätsknick. Die Produktivität sinkt, da die Mitarbeiter mit dem neuen System noch nicht vertraut sind und die Durchlaufzeit einzelner Prozesse steigt deutlich an. In dem Maße wie die Nutzung des neuen Systems zunimmt, steigt auch die Produktivität wieder an und erreicht ein Niveau, welches über dem der Ausgangsituation liegt. Erfahrungswerte zeigen, dass bei der Einführung klassischer CASE-Tools in der Regel die Zeitspanne bis zum Erreichen der ursprünglichen Produktivität bis zu einem Jahr beträgt, wobei eine Steigerung der Produktivität nicht erreicht wird. Dies bedeutet auch, dass Kosten für zukünftige Projekte nicht signifikant gesenkt werden.

Die eigentliche Auswirkung, die durch den Einsatz von CASE-Tools, bzw. einer CASE-Umgebung eintritt, ist eine Verbesserung der Qualität der erzeugten Software. Nach [9] werden 70% aller CASE-Tools bereits ein Jahr nach der Einführung nicht mehr genutzt, 25% werden nur von einer Entwicklergruppe innerhalb einer größeren Abteilung genutzt und nur 5% werden flächendeckend eingesetzt. Dies beruht nach Meinung des Verfassers zum einen auf der Komplexität dieser Systeme, zum anderen fällt auf, dass die Mehrzahl der CASE-Tools ihren Schwerpunkt in der Optimierung des Quelltextes haben. Sie stellen sicher, dass eine geschlossene Kette, ausgehend von einer formalen objektorientierten Analyse, mit zugehöriger Dokumentation in Form von Diagrammen etc. über das Design bis hin zur Codierung erstellt wird. Bei der Betrachtung der gestellten Anforderungen und der Ergebnisse der Befragungen in den Software-Unternehmen werden diese Aspekte jedoch in der Praxis nicht als kritisch gesehen.

Eine abschließende Bewertung der CASE-Tools im Zusammenhang mit dem SLM zeigt, dass zwar die verfolgten Ziele teilweise identisch sind, wie beispielsweise eine generelle Verbesserung der Qualität und Wartungsfreundlichkeit eines Softwaresystems, dass jedoch in der Praxis der Lösungsansatz, der von der Mehrzahl der CASE-Tools verfolgt wird, nicht zur Erreichung dieser Ziele führt. Das SLM kann vielmehr auch als eine neue Art von CASE-Umgebung interpretiert werden.

4.5 Neue Technologien der Software-Entwicklung

Mit der Entwicklung der neuen grafischen Betriebssysteme wie Windows im Bereich der PC oder X-11 im Bereich der Unix-Workstations hat die Komplexität von Computerprogrammen enorm zugenommen. Gemäß dem lange gültigen EVA[9]-Prinzip, wurde früher bei der Programmierung der Schwerpunkt auf die Umsetzung eines Algorithmus für eine bestimmte Problemlösung gelegt. Mit dem Aufkommen der grafischen Benutzeroberflächen stiegen jedoch die Anforderungen an die Benutzeroberfläche eines Computerprogramms. Heute sind interaktive Computerprogramme, die dem Benutzer die Möglichkeit eines flexiblen Dialogs bieten, Stand der Technik. Ebenso gewinnen andere Aspekte neben der reinen Umsetzung eines Algorithmus, wie beispielsweise die Ablage der Daten in einer Standard-Datenbank an Bedeutung. Mit der zunehmenden Komplexität der Computerprogramme haben sich auch neue Entwicklungstechniken etabliert, die mehrheitlich aus dem Prinzip der objektorientierten Software Entwicklung entstanden sind.

Das Ziel der Software Entwicklungstechnologien besteht in erster Linie darin, eine modulare Software-Entwicklung zu ermöglichen. Die Modularisierung zur Entwicklungszeit eines Computerprogramms wird in der Regel durch die Verwendung einer objektorientierten Programmiersprache ermöglicht. Diese bietet mit Hilfe von vererbbaren Klassen die Möglichkeit, den Quelltext eines Programms sauber in verschiedene Module aufzuteilen, wodurch auch die Wiederverwendung einmal erstellter Klassen für andere Computerprogramme gegeben ist. Wünschenswert ist jedoch darüber hinaus eine Modularisierung auch zur Ausführungszeit. Das heißt, dass fertige, lauffähige Module nach dem Baukastenprinzip miteinander kombiniert werden können. Für diese Problemstellung haben sich mehrere Technologien entwickelt, die im Folgenden näher untersucht werden, um die für das SLM am besten geeignete Technologie herauszufinden.

Der erste Ansatz bestand darin, Funktionalitäten eines Software-Moduls in Form von sogenannten Bibliotheken zusammen zu fassen, welche zur Laufzeit von der ausführbaren

[9] EVA: Eingabe – Verarbeitung - Ausgabe

Programmdatei hinzugeladen werden. Im Windows-Umfeld heißen diese Bibliotheken DLL[10], im Bereich der Unix-Betriebsysteme wird von „Shared Libraries“ gesprochen. Die Verwendung von DLL’s hat sich im Laufe der Zeit sehr großer Beliebtheit erfreut, selbst das Betriebssystem Windows besteht zu einem großen Teil aus DLL’s. Sehr schnell zeigten sich jedoch auch gravierende Probleme bei dieser Variante. Die Vielzahl der unterschiedlichen DLL’s, welche auf einem durchschnittlichen Rechner vorhanden sind, ist nicht mehr durchschaubar. Erschwerend kommt hinzu, dass es im Hinblick auf die Versionen von DLL’s erhebliche Probleme gibt.

In der Regel verwendet jedes Windows-Programm mehrere System-DLL’s. Entsprechend werden bei jeder Installation eines neuen Anwendungsprogramms auf einem Rechnersystem auch unter Umständen solche allgemeingültigen DLL’s installiert. Es ist nicht ungewöhnlich, dass ein anderes Anwendungsprogramm jedoch nur mit der zuvor vorhandenen Version dieser DLL einwandfrei funktioniert. Bei der nächsten Ausführung dieses Programms kommt es demnach zu einem Fehler, der gewöhnlich zum Absturz des Programms führt. Dass die Ursache für den Absturz des Programms, welches bis dahin immer einwandfrei funktionierte, die Installation eines anderen Programms ist, kann der Anwender im Normalfall nicht mehr erkennen. Die Verwendung von dynamischen Bibliotheken ist also mit schwerwiegenden Nachteilen verbunden. Diese Nachteile sollen mit Hilfe von COM[11] behoben werden.

Bei COM handelt es sich um ein Protokoll, welches von Microsoft veröffentlicht wurde. Dieses Protokoll definiert die Kommunikation verschiedener Softwaremodule. Im Vordergrund stehen hier eigenständige Komponenten, welche mit Hilfe von definierten Schnittstellen verbunden werden. Die zentrale Bedeutung kommt hierbei den Schnittstellen zu. Jeder COM-Server, also jede Komponente, die über eine COM-Schnittstelle Funktionalitäten exportiert, veröffentlicht seine Schnittstelle durch die sogenannte *Typelib*. Per Definition darf sich eine Schnittstelle niemals ändern. Mit Ausnahme der für die Kommunikation notwendigen COM-Funktionen selbst, enthält eine solche Schnittstelle keine statischen oder dynamischen Eintrittspunkte oder fest einprogrammierte Adressen. Die COM-Technologie stellt einen wesentlichen Schritt in Richtung auf ein objekt- und komponenten-

[10] DLL: Dynamic Link Library

[11] COM: Component Object Model

orientiertes System dar. Der Nachteil von COM ist jedoch, die Beschränkung auf die Windows Betriebssysteme im PC-Umfeld.

Ähnlich wie COM, existiert mit CORBA[12] ein vergleichbares Verfahren für die UNIX-Welt. Zwar wurden in der Folge mit DCOM und COM+ Erweiterungen des COM-Protokolls angeboten, welche die Hardware- und Betriebssystemgrenzen überwinden sollen, diese haben sich jedoch in der Praxis nicht etabliert. Die neueste Technologie wird unter der Bezeichnung „.NET“[13] von Microsoft veröffentlicht. An der Definition dieses neuen Standards sind neben Microsoft auch andere namhafte Hersteller wie IBM, SUN etc. beteiligt. .NET hat das Ziel eine hardwareunabhängige und betriebssystemunabhängige Grundlage für internetbasierte Software bereitzustellen. Die Plattform .NET wird in Form des .NET-Frameworks von Microsoft vertrieben. Nach Angaben von Microsoft erfüllt das .NET-Framework die folgenden Ziele:

- Bereitstellung einer konsistenten, objektorientierten Programmierumgebung, in der Objektcode gespeichert wird. Die Ausführung erfolgt dann entweder lokal oder über Remotezugriff bzw. lokal mit Verteilung über das Internet.

- Bereitstellung einer Codeausführungsumgebung, mit der Konflikte bei der Softwareweitergabe und Versionskonflikte auf ein Minimum beschränkt werden.

- Bereitstellung einer Codeausführungsumgebung, die eine sichere Ausführung gewährleistet und zwar auch von Code, der von unbekannten oder nur halb-vertrauenswürdigen Dritten erstellt wurde.

- Bereitstellung einer Codeausführungsumgebung, die nicht mehr die bei interpretations- oder skriptbasierten Umgebungen auftretenden Leistungsprobleme aufweist.

[12] CORBA: Common Object Request Broker

[13] .NET: sprich: „Dot-Net“

- Schaffung einer konsistenten Entwicklungsumgebung für die verschiedensten Anwendungsarten, wie beispielsweise Windows- und webbasierte Anwendungen.

- Aufbau der gesamten Kommunikation auf Industriestandards, um die Integration von Code, der auf .NET-Framework basiert, in jedem anderen Code zu gewährleisten.

Das .NET-Framework stellt also die Basis für eine vereinfachte und beschleunigte Entwicklung von Software bereit, wobei die häufigsten Ursachen für Programmfehler konzeptionell bereits ausgeschlossen werden. Das Framework besteht im wesentlichen aus zwei Bereichen; der „Common Language Runtime“ (CLR) [16] und der Klassenbibliothek des .NET-Framework. Dabei tritt die CLR zur Ausführungszeit eines Programms als Verwaltungs- und Steuerungseinheit auf. Die CLR stellt Dienste für die Codeverwaltung, die Verwaltung verschiedener Threads und den Remote Zugriff zur Verfügung. Ebenso wird der von einem Programm angeforderte Speicher von der CLR verwaltet. Diese übergeordnete Speicherverwaltung (Garbage Collection) macht eine Verweiszählung innerhalb des Anwendungsprogramms überflüssig und trägt wesentlich zur Stabilität eines Computerprogramms bei [16].

Eine Vielzahl von Fehlerquellen in C / C++ Programmen beruht bekanntermaßen auf Problemen der Speicherverwaltung. Daneben erzwingt die CLR gewisse Richtlinien, die ein Programm erfüllen muss, wie beispielsweise Typsicherheit und Codegenauigkeit. Alle diese Eigenschaften der CLR dienen der Robustheit und Sicherheit eines Programms. Die CLR kennt grundsätzlich zwei Arten von Programmcode, zum einen den verwalteten Code (Managed Code), der innerhalb der CLR ausgeführt wird und den unverwalteten Code (Unmanaged Code), welcher außerhalb der CLR, auf herkömmliche Art ausgeführt wird. Die Präferenz liegt bei dem verwalteten Code, jedoch müssen bestimmte Programme, wie beispielsweise Hardware-Treiber als unverwalteter Code ausgeführt werden. Bei der Verwendung von verwaltetem Code, wird ausgehend vom Quelltext zunächst ein Compilat in der IL (Intermediate Language) erzeugt [17]. Dieser IL-Code wird zur Ausführungszeit durch einen „Just In Time“-Compiler in echten Maschinencode übersetzt, ähnlich wie ein Java-Programm, welches in einer Virtuellen Maschine ausgeführt wird. Auf diese Art ist eine hardwareneutrale Programmierung möglich, wobei jedoch sowohl die von den

Interpretersprachen bekannten Probleme ausgeschlossen werden, als auch eine Performance erreicht wird, die mit der von herkömmlichen C / C++ Programmen vergleichbar ist.

Ein anderer wichtiger Bereich der CLR ist die Sicherstellung der Typsicherheit. Dies ist vor allem bei der Verwendung mehrerer Programmiersprachen innerhalb eines Programmsystems von entscheidender Bedeutung. So kann sichergestellt werden, dass eine Variable vom Typ Zeichenkette in der Programmiersprache Visual Basic .NET genauso definiert ist, wie in Visual C++ .NET. Gerade diese Unabhängigkeit von einer speziellen Programmiersprache ist einer der größten Vorteile des .NET-Frameworks. Dabei geht die Kompatibilität unter den Programmiersprachen noch weiter. Bei der Verwendung von Modulen innerhalb eines Gesamtsystems können problemlos Module miteinander verbunden werden, die in unterschiedlichen Sprachen programmiert wurden. Dabei können beispielsweise von Klassen, die in einem Modul in einer Programmiersprache definiert wurden, in einem anderen Modul sowohl Objekte instanziiert als auch weitere Klassen abgeleitet werden. Zur Unterstützung dieser Kompatibilität enthält das .NET Framework mehrere neue Programmiersprachen. Dazu zählt zum einen Visual Basic .NET. Dabei handelt es sich um eine Weiterentwicklung von Visual Basic mit den objektorientierten Elementen, die erforderlich sind, um die oben genannten Eigenschaften zu erfüllen.

Diese Sprache richtet sich vordringlich an Software Entwickler, die bisher mit Visual Basic gearbeitet haben. Mit der Programmiersprache C#[14] hat Microsoft außerdem eine weitere neue Sprache eingeführt, die als Weiterentwicklung von C / C++ gesehen wird [17]. Diese Sprache bietet eine starke Typsicherheit, wie auch C / C++, bietet darüber hinaus aber die bereits angesprochene Garbage Collection. Zusammengefasst bietet C# eine ähnliche Syntax wie C++ und ermöglicht einen komplett objektorientierten Ansatz, wobei die Leistungsfähigkeit von C++ mit dem einfachen Aufbau von Basic kombiniert wurde. Für die Verwendung des .NET-Frameworks ist jedoch die Wahl der Programmiersprache nicht relevant. Neben den in der Entwicklungsumgebung bereits enthaltenen Sprachen, wie Basic, C/C++, C# etc. sind auch Compiler von Drittanbietern beispielsweise für COBOL oder EIFFEL erhältlich.

[14] sprich „C-Sharp“

Den zweiten Bereich innerhalb des .NET-Frameworks neben der CLR bildet die .NET Klassenbibliothek. Dabei handelt es sich um eine umfangreiche Sammlung von Klassen für die verschiedenen Aufgabenbereiche, wie beispielsweise den Zugriff auf eine Datenbank, Elemente der Benutzeroberfläche oder die Verwaltung von Datentypen. Kennzeichnend für die .NET Klassenbibliothek ist, dass hier ebenfalls die Interopabilität zwischen den unterschiedlichen Programmiersprachen gewährleistet ist. Dies bedeutet zum Beispiel, dass die Windows-Steuerelemente für einen Benutzerdialog, die von einem Basic-Programm verwendet werden identisch sind mit denen, die von einem C#-Programm verwendet werden. Im einzelnen enthält die Klassenbibliothek neben den zahlreichen Systemklassen, die in einem Anwendungsprogramm verwendet werden können, auch Schnittstellen und Werttypen. Da gerade die Werttypen für die Interopabilität eine besondere Bedeutung haben, da sie die Grundlage für den Entwurf neuer Komponenten darstellen, werden im folgenden die Funktionen dieser Typen aufgeführt.

- Darstellung von Basisdatentypen und -ausnahmen
- Kapseln von Datenstrukturen
- E/A-Operationen
- Zugriff auf Informationen über geladene Typen
- Aufrufen von .NET-Framework-Sicherheitsüberprüfungen
- Datenzugriff und Bereitstellen einer eigenständigen Client-GUI sowie einer servergesteuerten Client-GUI

Die folgende Tabelle gibt einen Überblick über die Basisdatentypen und ihre entsprechende Darstellung in den jeweiligen Programmiersprachen:

Kategorie	Klassenname	Beschreibung	Datentyp in Visual Basic	Datentyp in C#	Datentyp in den verwalteten C++-Erweiterungen	Datentyp in JScript
Ganze Zahl	Byte	Eine 8-Bit-Ganzzahl ohne Vorzeichen.	Byte	byte	char	byte
	SByte	Eine 8-Bit-Ganzzahl mit Vorzeichen. Nicht CLS-kompatibel.	SByte Kein integrierter Typ.	sbyte	signed char	SByte
	Int16	Eine 16-Bit-Ganzzahl mit Vorzeichen.	Short	short	short	short
	Int32	Eine 32-Bit-Ganzzahl mit Vorzeichen.	Integer	int	int - oder - long	int
	Int64	Eine 64-Bit-Ganzzahl mit Vorzeichen.	Long	long	__int64	long
	UInt16	Eine 16-Bit-Ganzzahl ohne Vorzeichen. Nicht CLS-kompatibel.	UInt16 Kein integrierter Typ.	ushort	unsigned short	UInt16
	UInt32	Eine 32-Bit-Ganzzahl ohne Vorzeichen. Nicht CLS-kompatibel.	UInt32 Kein integrierter Typ.	uint	unsigned int - oder - unsigned long	UInt32
	UInt64	Eine 64-Bit-Ganzzahl ohne Vorzeichen. Nicht CLS-kompatibel.	UInt64 Kein integrierter Typ.	ulong	unsigned __int64	UInt64
Gleitkomma	Single	Eine Gleitkommazahl einfacher Genauigkeit (32 Bit).	Single	float	float	float
	Double	Eine Gleitkommazahl doppelter Genauigkeit (64 Bit).	Double	double	double	double
Logisch	Boolean	Ein boolescher Wert (true oder false).	Boolean	bool	bool	bool
Sonstige	Char	Ein Unicode-Zeichen (16 Bit).	Char	char	wchar_t	char
	Decimal	Ein 96-Bit-Dezimalwert.	Decimal	decimal	Decimal	Decimal
	IntPtr	Eine ganze Zahl mit Vorzeichen, deren Größe von der zugrunde liegenden Plattform abhängt (32-Bit-Wert auf einer 32-Bit-Plattform und 64-Bit-Wert auf einer 64-Bit-Plattform).	IntPtr Kein integrierter Typ.	IntPtr Kein integrierter Typ.	IntPtr Kein integrierter Typ.	IntPtr
	UIntPtr	Eine ganze Zahl, deren Größe von der zugrunde liegenden Plattform abhängt (32-Bit-Wert auf einer 32-Bit-Plattform und 64-Bit-Wert auf einer 64-Bit-Plattform). Nicht CLS-kompatibel.	**UIntPtr** Kein integrierter Typ.	**UIntPtr** Kein integrierter Typ.	**UIntPtr** Kein integrierter Typ.	**UIntPtr**
Klassenobjekte	Object	Der Stamm der Objekthierarchie.	**Object**	**object**	**Object***	**Object**
	String	Eine unveränderliche Zeichenfolge fester Länge mit Unicode-Zeichen.	**String**	**string**	**String***	**String**

Tabelle 4.1: Basisdatentypen der .NET-Klassenbibliothek

Da die .NET-Klassenbibliothek sehr umfangreich ist, soll an dieser Stelle auf eine Beschreibung der einzelnen Klassen verzichtet werden, die in der Dokumentation des .NET-Frameworks gefunden werden kann.

Ein weiterer Aspekt zur Beurteilung des .NET-Frameworks ist die Migration bereits vorhandener Komponenten, die beispielsweise mit Hilfe der Programmiersprache Visual Basic oder Visual C / C++ erstellt wurden. Gemäß der .NET-Dokumentation ist dies zwar möglich, die Erfahrungen des Verfassers zeigen jedoch, dass hier in der Praxis eine Neuentwicklung in der Regel die bessere Variante darstellt. Da jedoch für die Neuentwicklung des SLM selbst keine Rücksicht auf bereits vorhandene Komponenten genommen werden muss, hat dieser Aspekt im Rahmen der konkreten Aufgabenstellung nur eine untergeordnete Bedeutung.

4.6 Ausgewählte Technologien und Werkzeuge für das Konzept

Auch wenn das Konzept für das SLM allgemeingültig formuliert wird, muss bei der Formulierung der einzelnen Algorithmen die spätere Umsetzung berücksichtigt werden. Es ist also notwendig bereits jetzt die Technologieplattform sowie die verwendete Programmiersprache festzulegen, da sich diese auf die Architektur des SLM-Programmsystems auswirkt.

Als Basistechnologie für das Konzept des SLM wird das .NET-Framework verwendet. Wie bereits in dem vorhergehenden Kapitel beschrieben, bietet .NET bereits konzeptionelle Grundlagen für die Stabilität und Zuverlässigkeit eines Computerprogramms. Darüber hinaus ist die angestrebte Plattformunabhängigkeit durch die konsequente Unterstützung der Web-Technologie hier am einfachsten zu erreichen. Durch die enthaltene .NET-Klassenbibliothek stehen eine Vielzahl von Basisklassen für die benötigten Module des SLM zur Verfügung, wie für den Zugriff auf eine Datenbank oder die Realisierung einer Client / Server-Architektur. Dadurch kann schon bei der Entwicklung des SLM ein großer Teil der Basisfunktionalität wieder verwendet werden und muss nicht komplett neu implementiert werden. Durch die breit gefächerte Unterstützung verschiedener Programmiersprachen können außerdem die einzelnen Module des SLM von anderen Modulen, welche in einer

anderen Programmiersprache implementiert sind, problemlos verwendet werden. Dies ist vor allem sinnvoll, da es sich bei dem SLM um ein stark modularisiertes Gesamtsystem handelt, und da einige Module in andere Anwendungsprogramme integriert werden.

Zur Integration in ein externes Programm kann es jedoch notwendig sein, eine spezifische Programmiersprache zu verwenden. Bei einer einheitlichen Verwendung des .NET-Framework ist eine solche heterogene Landschaft jedoch ohne redundanten Code bzw. ohne redundante Schnittstellen möglich. Mit der Festlegung auf .NET als Grundlage für das Konzept des SLM, wird die Auswahl der verwendeten Programmiersprache erleichtert.

Auf Grund der gestellten Anforderungen kann nur eine Programmiersprache gewählt werden, die eine objektorientierte Entwicklung unterstützt. Mit „Visual Basic .NET“ (VB-.NET) stellt das .NET-Framework eine objektorientierte Programmiersprache zur Verfügung, die sowohl einen sicheren und strukturierten Aufbau der Programme gewährleistet und andererseits über die Einfachheit und Klarheit verfügt, die bereits von Visual Basic bekannt ist. An dieser Stelle muss nochmals darauf hingewiesen werden, dass es sich bei VB-.NET keineswegs nur um eine neue Version von Visual Basic handelt, sondern dass alle Konzepte des .NET-Frameworks konsequent in diese Programmiersprache integriert wurden. Damit ist VB-.NET als die primäre Programmiersprache für das SLM geeignet.

Es wurde bereits angesprochen, dass einige Module des SLM in andere Programme integriert werden müssen. Dies betrifft hauptsächlich die Schnittstellen des SLM zu anderen Programmen, die in den Softwarelebenszyklus involviert sind. Als einfaches Beispiel sei ein Texteditor zum Bearbeiten der Quelltexte genannt. Um die gewünschten Dateien mit Hilfe des SLM bearbeiten zu können und diese nach erfolgter Bearbeitung wieder an das SLM übergeben zu können, ist eine Integration eines Editors in das SLM notwendig. Diese Integration erfordert, dass Funktionen des SLM auch innerhalb des Editors zur Verfügung stehen. Diese müssen also in den Editor integriert sein. Eine solche Integration in ein externes Programm hängt in erster Linie davon ab, welche Möglichkeiten der Anbindung dieses Programm bietet. Viele Programme verfügen über eine Programmierschnittstelle, über die eine Erweiterung möglich ist. Mit der Nutzung einer solchen individuellen Programmierschnittstelle ist jedoch in der Regel auch eine spezielle Programmiersprache oder

eine individuelle produktspezifische Makrosprache verbunden. Es wird also im Rahmen der Schnittstellen des SLM zu anderen Systemen erforderlich sein, eine andere als die primäre Programmiersprache zu verwenden.

Dem SLM-System liegt, wie bereits besprochen, eine relationale Datenbank zu Grunde. Der Zugriff auf relationale Datenbanksysteme erfolgt über die normierte Zugriffssprache SQL. Datenbankzugriffe über diese Sprache werden bei umfangreichen Datenbanken mit komplexen Abhängigkeiten sehr schnell unübersichtlich, insbesondere dann, wenn die Datenbank so gestaltet werden soll, dass zur Laufzeit Änderungen an der Tabellenstruktur vorgenommen werden sollen. Dies erfordert, wenn die Zugriffe auf die Datenbank dementsprechend flexibel sein sollen, eine aufwändige Programmierung. Eine mögliche Lösung für ein solches Problem ist z.B. der Einsatz einer Metadatenbankstruktur, also einer Beschreibung der Relationenstruktur der Datenbank in der Datenbank selber. Um dieses Problem in den Griff zu bekommen, wurden von Microsoft nacheinander verschiedene Technologien entwickelt, die den SQL-Zugriff kapseln und den Zugriff auf Datenbanken beliebiger Hersteller erlauben, ohne dass am Programmcode Anpassungen vorgenommen werden müssen.

Als erstes sei hier Microsoft ODBC[15] genannt, dessen wesentliche Intention die Abstraktion des Datenzugriffes von der verwendeten Datenbank war und welches auch bereits eine gewisse Kapselung der SQL-Zugriffssprache bereitstellte. ODBC ist eine Programmier-schnittstelle zum Zugriff auf relationale Datenbanksysteme und besitzt keinen objekt-orientierten Ansatz. Auch sind bei ODBC die Datenzugriffe im wesentlichen fest verdrahtet und die gesamte Programmierung unflexibel.

Der nächste Schritt Microsofts war das OLE[16]-DB, ein objektorientierter Ansatz, der auf der Microsoft-COM-Technologie aufsetzt. OLE-DB bietet eine weitere Stufe der Abstraktion von der Datenquelle. Es kann nun im wesentlichen auf alle tabellenorientierten Datenquellen zugegriffen werden, nicht nur auf SQL-Provider. Über OLE-DB kann der Programmierer nicht nur auf Datenquellen wie relationale Datenbanken zugreifen, sondern auch auf

[15] Open Database Connection

[16] Object Linking and Embedding

Tabellenkalkulationsdateien oder sogar entsprechend organisierte einfache Textdateien. OLE-DB ist eine Technologie, welche die Absicht hat, sämtliche möglichen Aspekte des Datenzugriffs abzubilden und ist dementsprechend komplex und unübersichtlich.

Um die Programmierung von Datenzugriffen für den Programmierer weiter zu vereinfachen, wurde ADO[17] entwickelt (vgl. Abbildung 4.1), welches wiederum die Funktionalitäten von OLE-DB kapselt, jedoch nicht sämtliche Möglichkeiten von OLE-DB bietet. ADO .NET ist die Weiterentwicklung von ADO im .NET-Standard und basiert deshalb nicht auf COM [21].

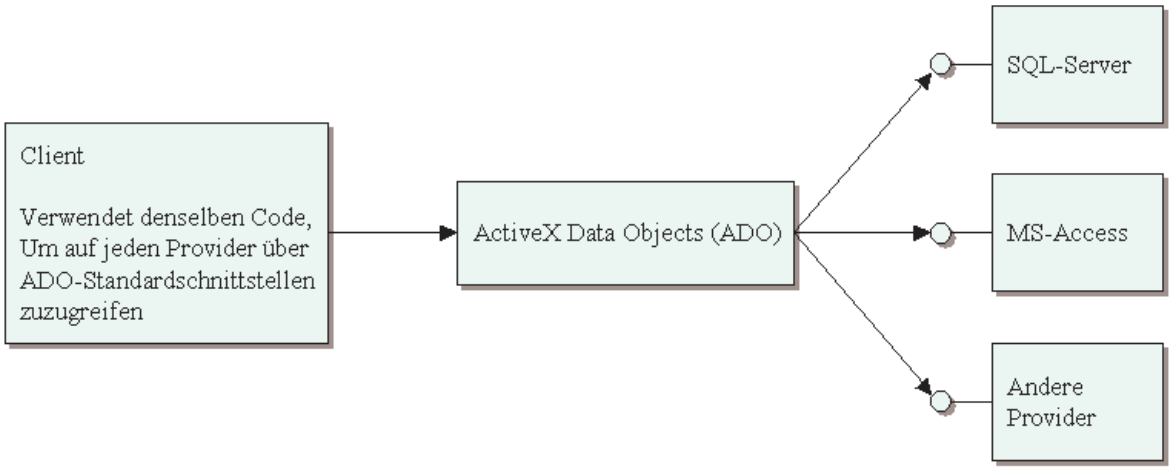


Abbildung 4.1: Abstraktion des Datenzugriffs durch ADO [21]

[17] Active-X Data Objects

5 Konzept für das SLM-System

Das SLM-System besteht aus verschiedenen Komponenten, die entweder auf einem Rechner installiert werden oder von verschiedenen Rechnern aus über ein Netzwerk miteinander kommunizieren. Aufgabe des SLM-Systems ist es, verschiedene CASE-Tools zu einer Entwicklungsumgebung zusammenzufügen, die Arbeit der Entwickler zu koordinieren und zu dokumentieren, sowie alle Informationen für Vertrieb und Support der entwickelten Software durchgängig verfügbar zu machen.

Das SLM-System ist aus diesem Grunde modular aufgebaut und besteht im wesentlichen aus den folgenden Komponenten:

- Relationale Datenbank: Speicherung aller Daten und Metadaten, sowie deren Beziehungen untereinander.
- SLM-Server Programm: Zugriff auf die Daten, den Lebenszyklus, den Workflow, die Benutzerverwaltung und Schnittstelle zum Client.
- Vault: Ablage der verwalteten Dateien.
- Client: Anmeldung am Server, Kommunikation zwischen Benutzer und Server, Schnittstelle zu den einzelnen CASE-Tools sowie sonstigen Programmen.

5.1 Aufbau des SLM-Systems

Da viele verschiedene Personen mit dem SLM-System gleichzeitig arbeiten, an vielen verschiedenen Orten und an vielen verschiedenen Rechnern, muss das System modular Aufgebaut sein. Hierbei existieren Funktionalitäten, die zentral auf einem Rechner lokalisiert sind und solche, die dezentral auf den einzelnen Arbeitsplatzrechnern der Anwender des Systems liegen, die aber mit den zentralen Funktionalitäten kommunizieren. Eine solche Architektur wird Client Server Architektur genannt [19]. Der zentrale Bestandteil des Systems ist hierbei das SLM-Server-Programm (vgl. Abbildung 5.1), welches für die Kommunikation zwischen der Datenbank, einer Standardkomponente, dem Vault und den einzelnen Clients verantwortlich ist, sowie für die gesamte Logik des Systems. Die Datenbank übernimmt die

Verwaltung der gesamten Metadaten [20] und der Vault die Verwaltung der Dateien, die im SLM-System abgelegt werden. Der Client schließlich ist für die Kommunikation des Benutzers mit dem Server verantwortlich, entweder direkt durch Aufruf der Benutzerschnittstelle des Clients oder über eine Integration des Clients in eine benutzerseitige Standardanwendung über eine entsprechende Schnittstelle. Sollen Dateien aus dem SLM-System dem Benutzer zur Verfügung gestellt werden, so macht der Client eine Anfrage an den Server über die entsprechende Datei und erhält als Antwort die Identifikation der Datei aus dem Vault. Nun kann der Client diese Datei aus dem Vault an die entsprechende Stelle des lokalen Dateisystems kopieren.

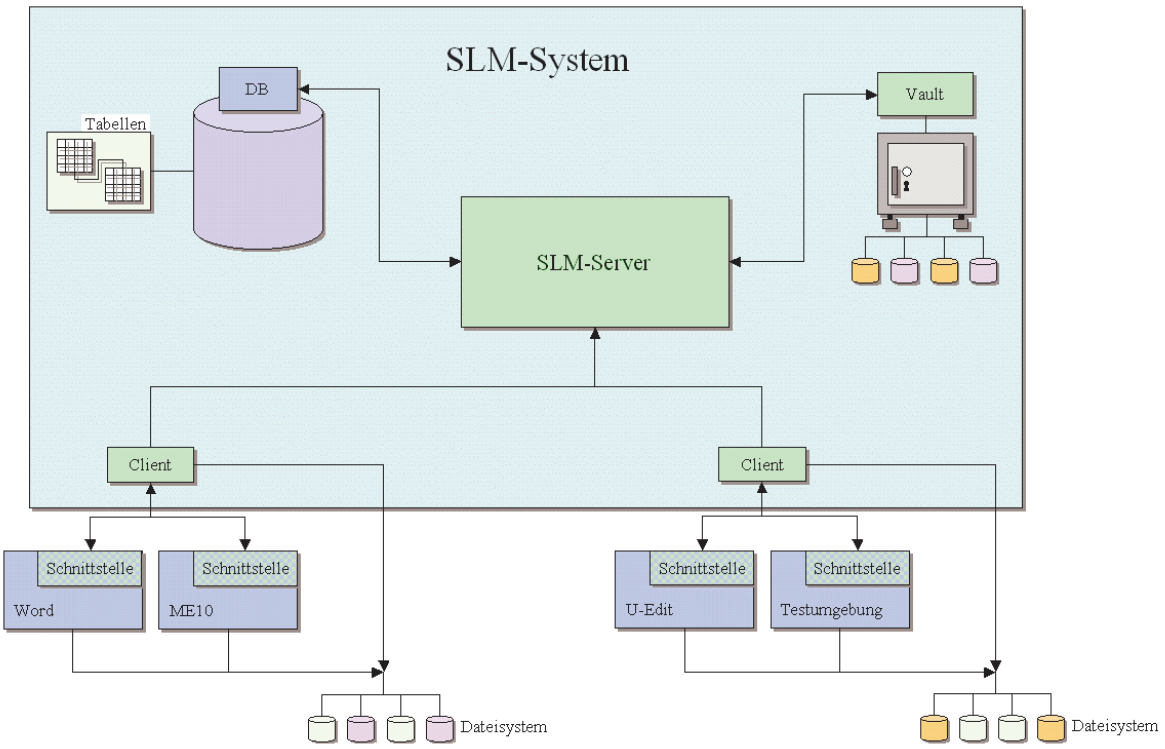


Abbildung 5.1: Aufbau des SLM-Systems

5.1.1 Client Server Architektur

Zentraler Bestandteil des SLM-Systems ist der Server-Dienst, der den Zugriff auf die Daten im SLM-System steuert. Der Zugriff auf die Datenbank und den Vault geschieht über Internetdienste, so dass Server-Dienst, Datenbank und Vault sich auf beliebigen Rechnern

befinden können. Auf den Rechnern der Anwender des SLM-Systems befindet sich jeweils ein Client, der die Kommunikation zwischen Anwender, seinen lokalen Anwendungen und dem Server-Dienst durchführt. Diese Kommunikation findet auf der Basis von exportierten Schnittstellen des Servers statt. Die Integration der Softwarekomponenten (CASE-Tools, Office-Programme, CAD-Programme) zu einer vollständigen spezifischen Entwicklungsumgebung erfolgt auf der Ebene des Clients, der hierzu die geeigneten Schnittstellen zur Verfügung stellt.

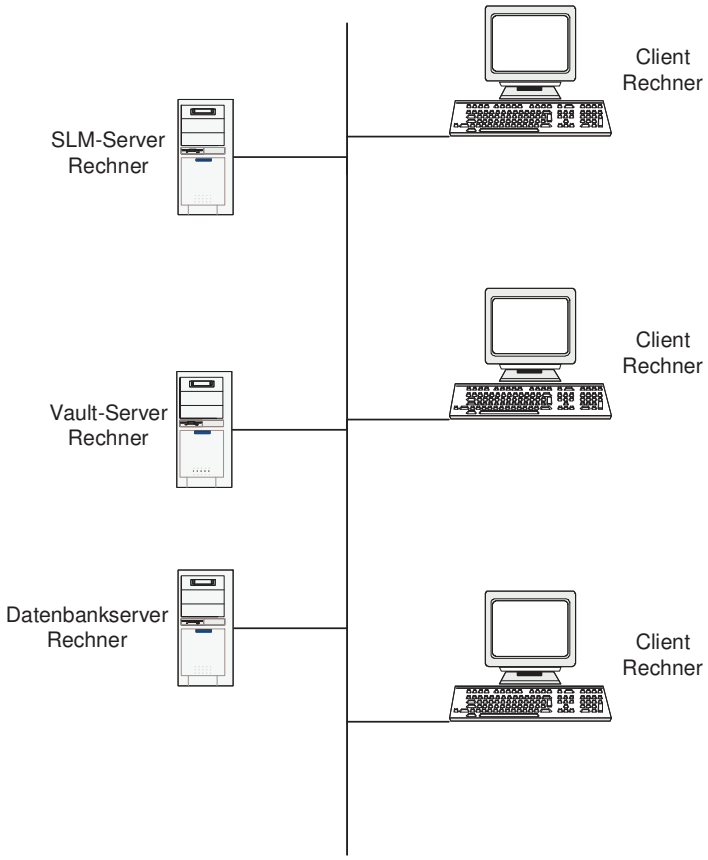


Abbildung 5.2: Netzwerkstruktur des SLM-Systems

5.1.2 Trennung von Datenhaltung und Prozesssteuerung

Die eigentliche Datenhaltung teilt sich in zwei Bereiche auf: die eigentliche Ablage der Dokumente in einem Vault und die Speicherung der Metadaten in einer Datenbank. In dieser Datenbank werden auch die Daten gehalten, die für die Prozesssteuerung notwendig sind. Zu

diesen Daten gehören der jeweilige Status einer Datei, die Mitarbeiter und deren Funktion, sowie die möglichen Statusänderungen. Die Prozesssteuerung wird über eine Software ausgeführt, welche die Daten in der Datenbank konsistent hält und für den Datenaustausch zwischen Vault und Client sorgt. Durch die Trennung wird erreicht, dass Prozesssteuerung, Vault und Datenbank jeweils auf verschiedenen Systemen arbeiten können, die dann auch jeweils einzeln gesichert und gewartet oder gegebenenfalls erweitert werden können (vgl. Abbildung 5.2).

5.1.3 Systemkommunikation / Datenaustausch

Die Ablage der Daten im SLM-System erfolgt in einem so genannten Vault. Dieser stellt im Prinzip ein flaches Dateisystem dar, also ein Dateisystem ohne tiefe Hierarchie, in dem die Dateien unter eineindeutigen Namen gespeichert werden. Die Vault-Namen der Dateien werden durch das SLM-System bzw. über Betriebsystemfunktionalitäten erzeugt. In der Datenbank werden zu diesen Dateien deren tatsächliche Namen, sowie ihr Platz in ihrer originären Verzeichnishierarchie verwaltet. Dies ist notwendig, da in unterschiedlichen Softwareprojekten bestimmte Dateinamen immer wieder vorkommen und somit verschiedene Versionen der gleichen Datei verwaltet werden müssen. Im Prinzip stellt der Vault eine Art chaotisches Lager[18] dar. Zur Vermeidung von Überläufen des Dateisystems, können beliebig viele verschiedene Vaults verwaltet werden, die auf unterschiedlichen Dateisystemen innerhalb des Netzwerkes existieren.

Der eigentliche Datenaustausch zwischen dem Client-Rechner und dem Server-Rechner erfolgt dann über eine sogenannte Freigabe. Dies ist ein Verzeichnis auf einem beliebigen Rechner, auf welches direkt über das Internet zugegriffen werden kann, wobei es für den zugreifenden Rechner so aussieht, als sei es ein lokales Verzeichnis. In dieses Verzeichnis werden, ausgelöst durch eine Anfrage des Clients an den Server, die angeforderten Dateien aus dem Vault hineinkopiert. Gegebenenfalls werden auch komplexe Verzeichnisstrukturen

[18] Automatische Hochregal-Lager größerer Betriebe werden oftmals als chaotische Lager ausgeführt. D.h. der Lagerort der gelagerten Güter ist mehr oder weniger zufällig und wird mit Hilfe eines Rechners verwaltet. Neue Güter werden automatisch im nächsten ausreichend dimensionierten freien Lagerplatz abgelegt.

erzeugt, wenn ganze Projekt-Verzeichnisse aus dem SLM-System angefordert werden. Den Dateien, die in Unterverzeichnissen einer Verzeichnisstruktur liegen, können hierbei bereits die originären Dateinamen gegeben werden. Da einzelne Dateien prinzipiell von mehreren Clients gleichzeitig für einen Lesezugriff angefordert werden können, müssen für diesen Zweck aus Gründen der Systemsicherheit, den Dateien und Verzeichnissen, die direkt in der obersten Ebene der Freigabe abgelegt werden, gesonderte Namen zugewiesen werden. Aus dieser Freigabe werden nach Benachrichtigung des Clients durch den Server diese Dateien bzw. Verzeichnisse auf den Client-Rechner verschoben und in ihre originären Namen umbenannt.

Die Übergabe von Dateien des Clients funktioniert auf die gleiche Weise. Es wird vom Client eine Anfrage an den Server für einen Namen gestellt, unter dem die entsprechende Datei oder das Verzeichnis in das Austauschverzeichnis kopiert wird.

5.2 Datenbankmanagementsystem

Zentraler Bestandteil des Systems ist ein relationales Datenbankmanagementsystem. In ihm werden alle Informationen gehalten, die zum Betrieb des SLM-Systems notwendig sind. Da das SLM-System sich problemlos in die bereits bestehende Software-Struktur eines Unternehmens einfügen muss, obwohl dort bereits ein relationales Datenbankmanagementsystem eingesetzt wird, muss es im wesentlichen unabhängig von dem jeweils vorhandenen Datenbank-System sein.

So kann z.B. in großen Softwarehäusern ein Oracle- oder ein MS-SQL-Server-System zum Einsatz kommen, um die notwendige Performance zu erreichen, während in einem kleinen Softwarehaus mit wenigen Arbeitsplätzen auch der Einsatz einer MySQL-Datenbank oder von MS-Access hinreichend ist. Ebenso wenig kommt aus diesem Grunde der Einsatz von Stored Procedures in Frage, da diese in Abhängigkeit vom verwendeten Datenbank-managementsystem gegebenenfalls neu entwickelt werden müssen.

5.2.1 Projektstruktur

Software wird im allgemeinen in der Struktur von Projekten entwickelt. Hierbei ist es gleichgültig, ob es sich dabei um spezielle Kunden-Software handelt oder um ein Software-Produkt, welches in einer festen Form allgemein vertrieben wird, immer werden die Daten einem spezifischen Projekt zugeordnet.

Dies ist auch an der generellen Vorgehensweise in der Softwareentwicklung zu sehen. Bereits bei der einfachen Erstellung eines neuen Projektes in MS-Visual Studio wird von dem Visual Studio ein spezielles Verzeichnis angelegt, welches den Namen der zu erstellenden Ausführungsdatei erhält, und welches im wesentlichen alle für die Erstellung der Software relevanten Informationen enthält. Hierzu gehören nicht nur die Quelldateien, sondern auch so genannte Ressourcen, in denen die Informationen über das Erscheinungsbild der zu erstellenden Anwendung enthalten sind. Dies sind z.B. Meldungen, die angezeigt werden oder auch Symbole für die grafische Benutzerschnittstelle. Eine weitere besondere Datei ist auch die Browser-Informations-Datei, die alle Informationen über die Zusammenhänge in den Quelldateien enthält. Diese kann aber jederzeit dynamisch aus den Quelldateien neu generiert werden.

Auch in der Art und Weise, wie die Software später auf den einzelnen Rechnern installiert wird, zeigt sich diese Struktur. Hierbei spielt es keine Rolle, unter welchem Betriebssystem (Windows, Unix, MacOS oder RiscOs) die Software installiert wird. Bis auf wenige Ausnahmen wird für die neu zu installierende Software ein neues Verzeichnis angelegt, in welchem idealerweise alle zum Betrieb der Software erforderlichen Informationen enthalten sind.

Die zentrale Organisationseinheit in der Softwareentwicklung ist also das Projekt, dem sich hierarchisch alle weiteren Daten und Dokumente zuordnen lassen, sowie die zur Verfügung stehenden Ressourcen, wie Mitarbeiter, Rechner, usw.. Beispielhaft wird hier die Verzeichnisstruktur eines Projektes aus dem CAD-Umfeld gezeigt (vgl.
Abbildung 5.3), bei dem es um die Erstellung einer komplexen Makroanwendung geht.

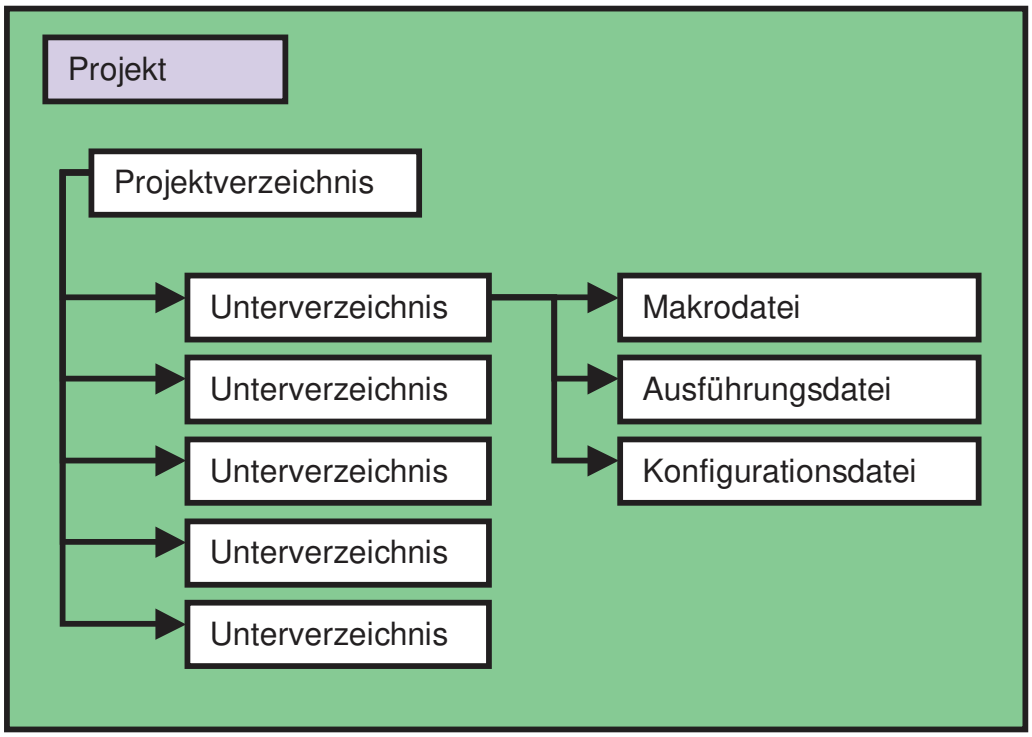


Abbildung 5.3: allgemeine (Verzeichnis-)Struktur eines Projektes

Als Basis für die Struktur der Datenbank wird das Projekt verwendet. Diesem Projekt werden entsprechende Ressourcen zugewiesen, wie z.B. Mitarbeiter, Rechner, usw.. Bei der Erstellung eines neuen Projekts werden vom SLM-System automatisch bestimmte Dokumente angelegt, ebenso beim Übergang von einer Phase in die nächste. Zu solchen Dokumenten gehören z.B. Pflichtenheft, Lastenheft, Anwendungsbeschreibung, usw. . Die Anlage dieser Dokumente muss frei konfigurierbar sein, ebenso wie die Phasenübergänge.

Die Zuordnung der einzelnen Entitäten[19] zueinander erfolgt über Links (Verknüpfungen). Prinzipiell existieren zwei unterschiedliche Arten von Links: Einfache Links, die eine einfache Zuordnung bedeuten und so genannte hierarchische Links, mit denen eine strukturelle Abhängigkeit beschrieben wird. Ein einfacher Link könnte so z.B. die Zuordnung einer Dokumentation zu einer Quellcodedatei sein, ein hierarchischer Link, die Zuordnung einer Quellcodedatei zu einem Projekt (vgl. Abbildung 5.4).

[19] Eine Entität ist ein individuelles, unterscheidbares und identifizierbares Exemplar von Dingen, Personen und Begriffen der realen oder der Vorstellungswelt [18].

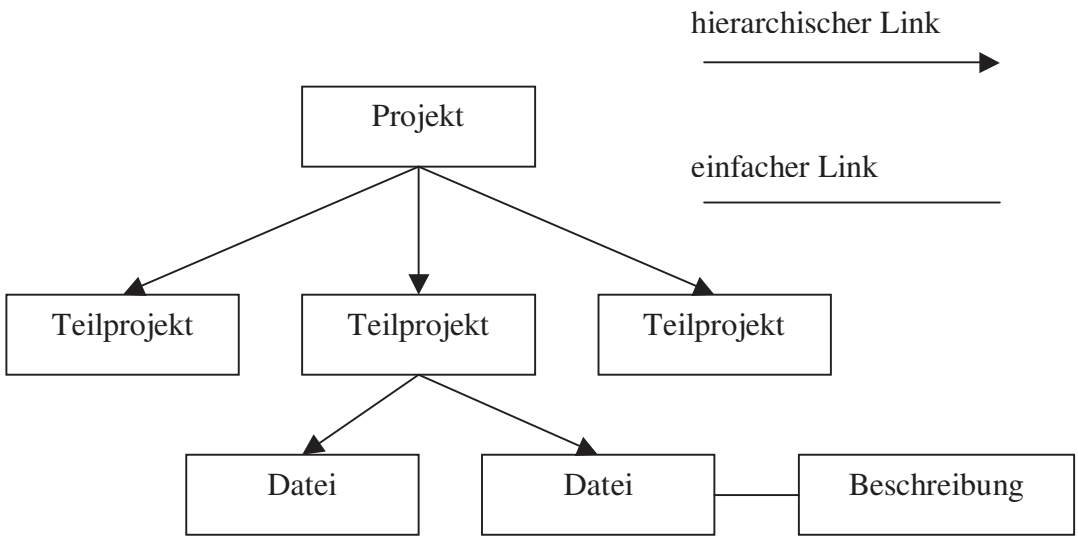


Abbildung 5.4: verschiedene Arten von Links

Um beliebige Entitäten miteinander „*verlinkbar*“ zu machen, muss sichergestellt werden, dass jedes Objekt eine eineindeutige Identifikation besitzt, und zwar unabhängig von der Tabelle, in der seine Daten und Metadaten abgelegt werden. Auch können in der Datenbank in Zukunft weitere Entitäten angelegt werden, die auch mit den bereits vorhandenen Entitäten verlinkt werden können.

Um dies sicherzustellen, existiert in der Datenbank eine spezielle Relation, die Objekt-Relation, in welcher alle Objekte der Datenbank einen Datensatz enthalten, mit einer eineindeutigen Objekt-ID. Diese Objekt-ID ist der Schlüssel dieser Relation. Die Datensätze in den objektspezifischen Relationen erhalten als Schlüssel die entsprechende Objekt-ID aus der Objekt-Relation. Um die Suche in den einzelnen Tabellen zu erleichtern, muss als weitere wichtige Information in der Objekt-Relation der Name der Relation abgelegt werden, in welcher die weiteren Datensätze zu einem spezifischen Objekt abgelegt sind.

5.2.2 Basis des Datenmodells

Das Datenmodell für das SLM-System muss in der Lage sein, die in 5.2.1 beschriebene Projektstruktur eines Softwareprojektes abzubilden. Ein Softwareprojekt setzt sich aus verschiedenen Objekten zusammen, die miteinander in Beziehung stehen. Zum einen gehören hierzu die eigentlichen Projektdaten, wie Quelltexte, die Ausführungsdateien, Konfigurationsdateien, dem Projekt zugeordnete Dokumentationen, sowie dem Projekt zugeordnete Ressourcen, wie Mitarbeiter, Rechner, usw.. Innerhalb eines Projekt existieren viele verschiedene Objekte. Diese Objekte werden in einer relationalen Datenbank durch Datensätze in Tabellen repräsentiert. Die Tabellen repräsentieren dabei so genannte Entitäten, die in etwa mit Objekt-Klassen gleichzusetzen sind. Hierbei gibt es im Prinzip zwei verschiedene Arten von Beziehungen zwischen diesen Entitäten:

- Einfache Zuordnungen und
- Hierarchische Zuordnungen, bei denen eine Entität von einer anderen abhängt.

Eine einfache Zuordnung wäre z.B. die Zuordnung eines Dokumentes zu einer Quelltextdatei oder einer Person zu einem Projekt. Hierarchische Zuordnungen sind Zuordnungen im Sinne von „*besteht aus*“ bzw. „*enthält*“, wie z.B. das Projekt besteht aus den Teilprojekten A und B; das Teilprojekt A hat das Verzeichnis X; das Verzeichnis X enthält die Dateien M und N usw. Die Zuordnungen von Informationen in einer relationalen Datenbank geschieht über den Dateninhalt, wofür Schlüssel verwendet werden. In Abhängigkeit von der Art der Beziehung werden entweder dem Objekt, welches mit genau einem anderen verknüpft ist, ein Attribut[20] mit der Identifikation des verknüpften Objektes hinzugefügt, oder aber es wird eine eigene Objekt-Klasse verwendet, welche als Attribute die Identifikationen der beiden zu verknüpfenden Objekte enthält. Über die erste Art lassen sich nur so genannte 1-N-Beziehungen realisieren, über die zweite Art auch beliebige beiderseitige Beziehungen, so genannte M-N-Beziehungen. Die Beziehungen verschiedener Objekte in der Datenbank muss somit über eine spezielle Entität, einer Link-Relation geschehen.

[20] Ein Entitätsattribut ist eine mit einem Namen versehene Menge an Fakten, die Aufgrund der Zuordnung von Eigenschaftswerten aus einer Menge zulässiger Eigenschaftswerte zu den Entitäten zustande kommt [19].

Da bei der Erstellung der Datenstruktur der Datenbank noch nicht klar ist, welche Entitäten bei zukünftigen Projekten wichtig sein können, kann es notwendig werden, die Datenbank um weitere Entitäten zu erweitern. Daher ist es notwendig, die Datenbank so zu gestalten, dass auch später der Datenbank hinzugefügte Entitäten, bzw. Objekt-Klassen ohne programmiertechnische Maßnahmen mit den anderen Objekten der Datenbank in Beziehung gesetzt werden können. Hierfür müssen in der Link-Entität die einzelnen Objekte so identifiziert werden, dass die Informationen später in den zugehörigen Tabellen schnell gefunden werden können. Im Prinzip sind zwei Arten der Informationsablage denkbar:

- Speicherung von Objekt- und Tabellenidentifikation beider verknüpfter Objekte oder
- nur Speicherung der Objektidentifikation, wobei sichergestellt sein muss, dass diese unabhängig von seiner Klasse eineindeutig für jedes Objekt sind.

Bei der ersten Art ist es also notwendig, bei der Suche nach einer Verknüpfung immer nach der Kombination von Klassen- und Objektidentifikation zu suchen. Dies kann bei einer sehr großen Link-Tabelle zu Performance-Problemen führen, da Maßnahmen zur Steigerung der Performance, hier in erster Linie die so genannte Indizierung einer Tabelle, welche die Suche in einer Tabelle durch Bisektion ermöglicht, nicht so gut greifen.

Daher wird für die Struktur der Datenbank eine für jedes Objekt eineindeutige Identifikation gewählt, die in einer gesonderten Tabelle verwaltet wird. Dies bedeutet gleichzeitig eine Abstraktion der in der Datenbank vorkommenden Entitäten zu einer Objekt-Entität, bei der auch sinnvolle und allen Entitäten der Datenbank gemeinsame Attribute abgelegt werden können. Durch diese Vorgehensweise wird auch zusätzlich eine Art Vererbung [15] in die Datenstruktur eingebracht, da nun alle Entitäten in der Relation von der Objekt-Entität abgeleitet sind. Über die dort vergebenen Objekt-Identifikation können dann auch die einzelnen Objekte in der Datenbank zueinander in Beziehung gesetzt werden.

Zusammenfassend lassen sich die Aufgaben, welche die Links in der Datenbank übernehmen, wie folgt definieren:

1. einfache Verknüpfung zweier Objekte, z.B. einer Quellcodedatei wird eine Dokumentation hinzugefügt.
2. hierarchische Verknüpfung von Objekten, z.B. einem Projekt wird eine Quellcodedatei hinzugefügt oder aber auch ein Teilprojekt.
3. Versionsverknüpfungen von Objekten. Dies geschieht bei jedem Eincheckvorgang eines Objektes, welches bereits in der Datenbank vorhanden ist.
4. Abbildung des Arbeitsflusses (Workflows): Verknüpfung von durchzuführenden Statusänderungen und den damit beauftragten Personen/Gruppen

5.2.3 Funktionsweise der Links

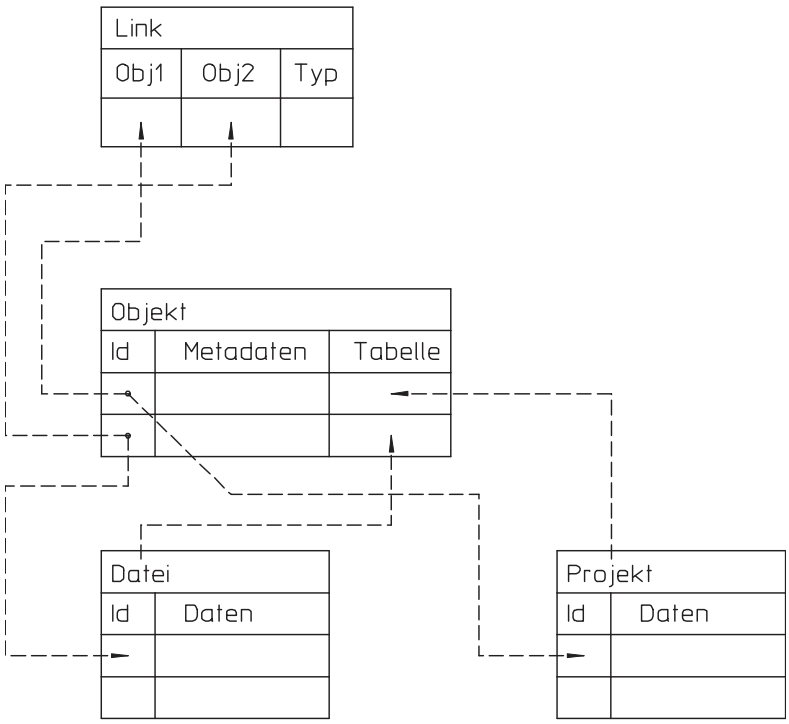


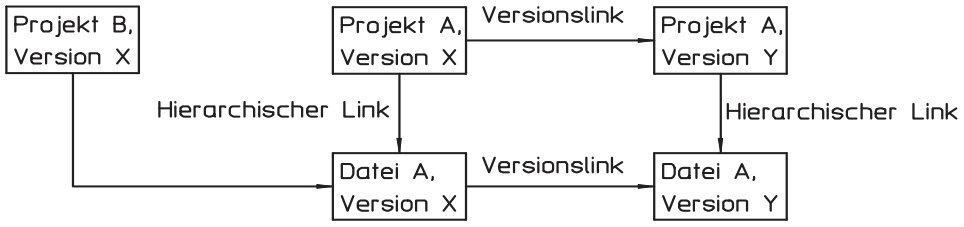
Abbildung 5.5: Objektstruktur und Verlinkung von Objekten

Die Art der Beziehungen der Objekte untereinander werden über den Typ des spezifischen Links geregelt. Das Auffinden von Objekten in der Datenbank geschieht durch die eineindeutige Objekt-Identifikation. Zur Erleichterung des Auffinden wird als Attribut eines Objekts noch die zugehörige Tabelle gespeichert, da sonst alle Tabellen der Datenbank sequentiell nach einer spezifischen Objekt-Identifikation durchsucht werden müssten. Eine

Vererbung in der Objekt-Struktur der Datenbank kann durch einfaches Anfügen eines Tabellenattributes an jede Tabelle, wie in der „Objekt“ Entität, erreicht werden, welches den Verweis auf die Eltern-Tabelle enthält. Dieses verweist auf eine weitere Tabelle, in der sich zusätzliche Informationen zu diesem Objekt befinden. Auf diese Weise wird ein Vererbungsmechanismus unter den Entitäten ermöglicht (vgl. Abbildung 5.5).

5.2.4 Versionisierung

Im Laufe eines Entwicklungsprozesses werden in der Regel von jedem in der Datenbank vorhandenen Objekt mehrere Versionen erzeugt. Bei jedem Eincheckvorgang wird die neue Version der jeweiligen Datei unter einem neuen Namen in den Vault übernommen und in der Datenbank wird ein neues Objekt erstellt, welches diese Datei repräsentiert. Die alten Versionen des Objektes und der ihm zugeordneten Dateien bleiben aber vorhanden, sofern sie nicht explizit aus der Datenbank gelöscht werden. Die Verknüpfung der verschiedenen Versionen eines Objektes untereinander erfolgt über einen speziellen Link-Typ, dem Versionslink. Mittels dieser Links werden die verschiedenen Versionen eines speziellen Objekts in einer Art verketteter Liste gehalten, so dass immer von einem Objekt die Vorgänger- und auch die Nachfolgeversion ermittelt werden kann. Auf diese Weise kann von jedem Objekt in der Datenbank eine Versionshistorie abgefragt werden (vgl. Abbildung 5.6).



Projekt
Id
1
2
3
...

Link			
Id1	Id2	Typ	Bez.
1	2	Version	B
4	5	Version	Y
1	4	Hierar.	
2	5	Hierar.	
3	4	Hierar.	

Datei
Id
4
5
6
...

Abbildung 5.6: Versionisierung von Objekten mit zugehöriger Tabellenstruktur

Bezeichnung und Ausgangsversion der Objekt (Datei und Tabelle) sind jeweils in der zugehörigen Objekt-Tabelle zu finden.

Sind Objekte mit einem anderen Objekt hierarchisch oder auf andere Weise verknüpft, müssen diese Verknüpfungen automatisch beim Eincheckvorgang aktualisiert werden. Dies geschieht folgendermaßen:

1. Identifikation des ursprünglichen Objekts in der Datenbank.
2. Erstellung eines Versionslink zwischen dem ursprünglichen Objekt, das die Vorgängerversion der Datei repräsentiert und dem neu angelegten Objekt.
3. Kopie aller allgemeinen Links, die sich auf die Vorgängerversion des Objektes beziehen, wobei in den kopierten Links die Identifikation der ursprünglichen Version des Objektes durch die Identifikation des neu angelegten Objektes ersetzt werden.
4. Löschen aller hierarchischer Links, die im Projektbezug auf die Vorgängerversion des Objektes verweisen.

Bei Schritt 3 ist darauf zu achten, dass zwar die allgemeinen Links, jedoch auf keinen Fall auch die hierarchischen Links auf das neue Objekt übertragen werden. Ein automatisches Kopieren der hierarchischen Links hätte den Effekt, dass eine neue Version eines Objektes, die im Zusammenhang mit einem speziellen Projekt erstellt wird, auch automatisch die Vorgängerversionen des Objektes in allen anderen Projekten ersetzen würde. Eine solche Veränderung der Projektstruktur darf jedoch unter keinen Umständen automatisch durch das System ohne Bestätigung durch den Benutzer erfolgen, da hierdurch auch unbeabsichtigte Seiteneffekte auftreten können. Es dürfen also nur diejenigen hierarchischen Links aktualisiert werden, die im Zusammenhang mit dem bearbeiteten Projekt stehen. Dies bedeutet, dass bei hierarchischen Links auch der Projektkontext gespeichert werden muss. Innerhalb eines Projektes genügt der Dateiname zusammen mit der Pfadangabe zur eindeutigen Identifikation eines Objektes. Werden zu einem späteren Zeitpunkt diese anderen Projekte bearbeitet, die diese spezielle Datei ebenfalls verwenden, so muss das SLM-System automatisch darauf hinweisen, dass bereits eine neuere Version dieser speziellen Datei existiert. Im Einzelfall kann nun der Anwender prüfen, ob diese neue Version für das Projekt verwendet werden soll oder nicht.

Wird von einem Objekt, welchem in der Datenbank andere Objekte zugeordnet sind, eine neue Version angelegt, so müssen alle Objekte in ihrer alten Version, die mit diesem Objekt verknüpft sind, ebenfalls mit diesem Objekt verknüpft werden. Die Verknüpfungen mit dem jetzt veraltetem Objekt jedoch bleiben bestehen.

5.2.5 ER-Diagramm der Datenbank

Die Abbildung des beschriebenen Datenmodells mit Hilfe einer relationalen Datenbank erfolgt auf Basis der gängigen Datenbank-Entwurfsmethoden [22]. Dementsprechend ergibt sich als Grundlage für die Implementierung der Datenbank das dargestellte Entity-Relationship-Model [13] (vgl. Abbildung 5.7). Für die Erstellung wurde in diesem Fall das Datenbankmanagementsystem MS-Access der Firma Microsoft verwendet.

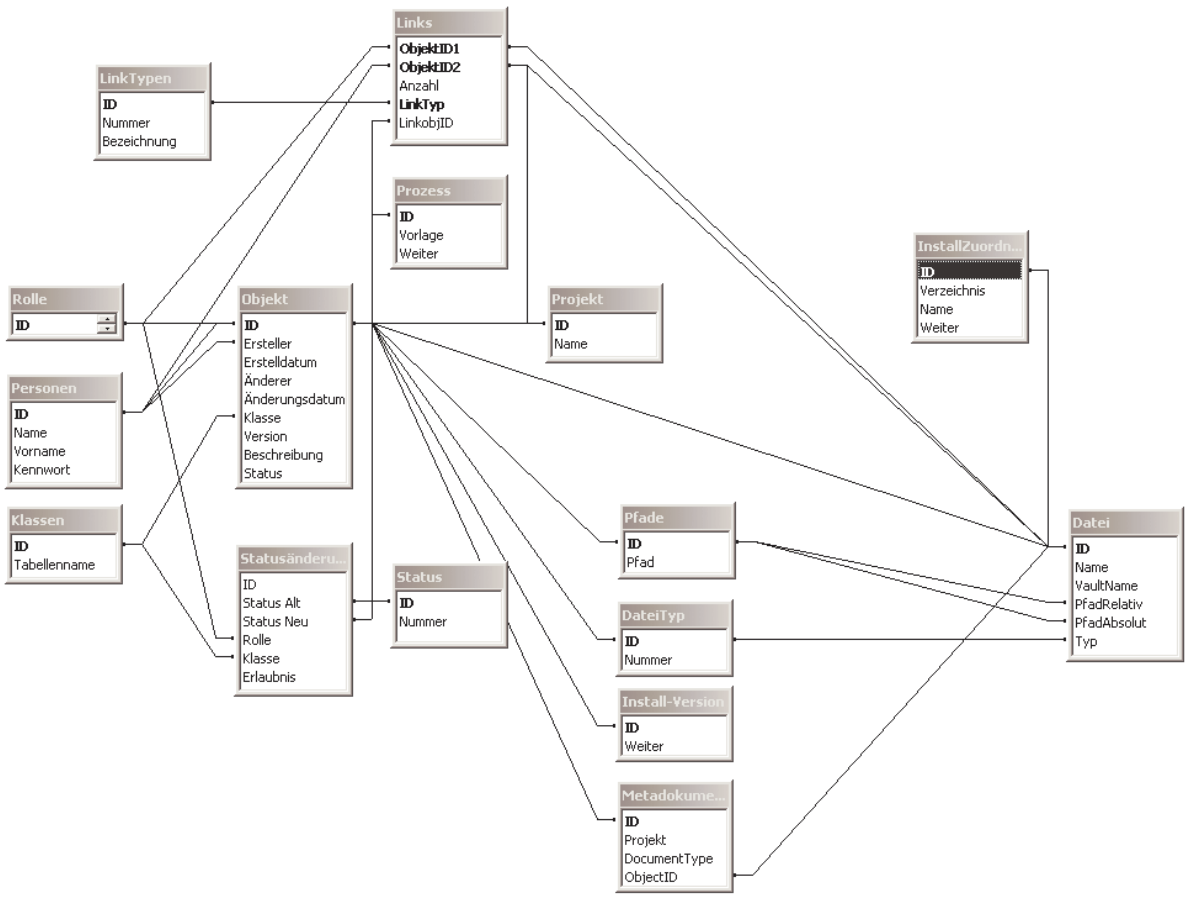


Abbildung 5.7: Darstellung der Datenbankstruktur

5.3 Server

Die Server-Anwendung des SLM-System läuft auf dem Server-Rechner als Dienst. Dies bedeutet, die Server-Anwendung läuft unabhängig von eventuell angemeldeten Benutzern im Hintergrund und ist immer bereit für die Annahme von Anfragen, ohne dass ein Benutzer das SLM-System explizit starten muss. Der Server zeichnet als wichtigste Aufgabe für die An- und Abmeldung von Sitzungen der Clients verantwortlich. Da die Anmeldung gegebenenfalls über ein öffentliches Netzwerk geschieht, muss sichergestellt sein, dass die Datenübertragung verschlüsselt geschieht. Die Rechtmäßigkeit der Anmeldung wird über eine in das SLM-System integrierte Benutzer- und Gruppenverwaltung geregelt. Weiterhin werden die Rechte des über einen Client angemeldeten Benutzers bei Zugriffen auf die Datenbank und die Clients überprüft. Hierzu gehört die Überprüfung, ob ein bestimmter Benutzer, der über einen Client angemeldet ist, eine bestimmte Transaktion bzw. Statusänderung durchführen, eine neue Datei anlegen oder ansehen darf.

Als weitere Hauptaufgabe kapselt und steuert der Server alle Zugriffe auf die Daten gegenüber den Clients. Zugriffe auf die Datenbank und den Vault werden durch den Server-Dienst gekapselt. Über seine Schnittstellen stellt der Server den Clients bestimmte Transaktionen zur Verfügung, die den Zugriff auf die Daten steuern, so dass keine Inkonsistenzen entstehen können. Eine weitere wichtige Aufgabe des Servers ist das automatische Aktualisieren der Abhängigkeiten zwischen den einzelnen Objekten in der Datenbank beim Einchecken neuer Versionen unter Beachtung des Projektkontextes.

Beim Starten des Server-Rechners muss das SLM-System automatisch gestartet werden, um sofort nach dem Hochfahren für Anfragen bereitzustehen. MS-Windows 2000 und XP sind genauso wie z.B. verschiedene UNIX-Derivate multitasking-, multiuser- und multisessions-fähige Betriebssysteme. Jeder Windows 2000 / XP Server hat auch die Fähigkeit als Terminalserver zu funktionieren, jedoch sind in der Regel die notwendigen Lizenzen nicht vorhanden, bis auf zwei Administrationslizenzen. Um nun den SLM-Server in Form einer Anwendung zu starten, müsste sich explizit ein Benutzer anmelden und angemeldet bleiben, bis der Server beendet wird. Daher wird der SLM-Server als Dienst implementiert, um unabhängig von den angemeldeten Benutzern zu sein. Beim Start des Systems wird der Dienst automatisch vom Betriebssystem initialisiert. Der schematische Ablauf als Flussdiagramm dargestellt ist wie folgt (vgl. Abbildung 5.8):

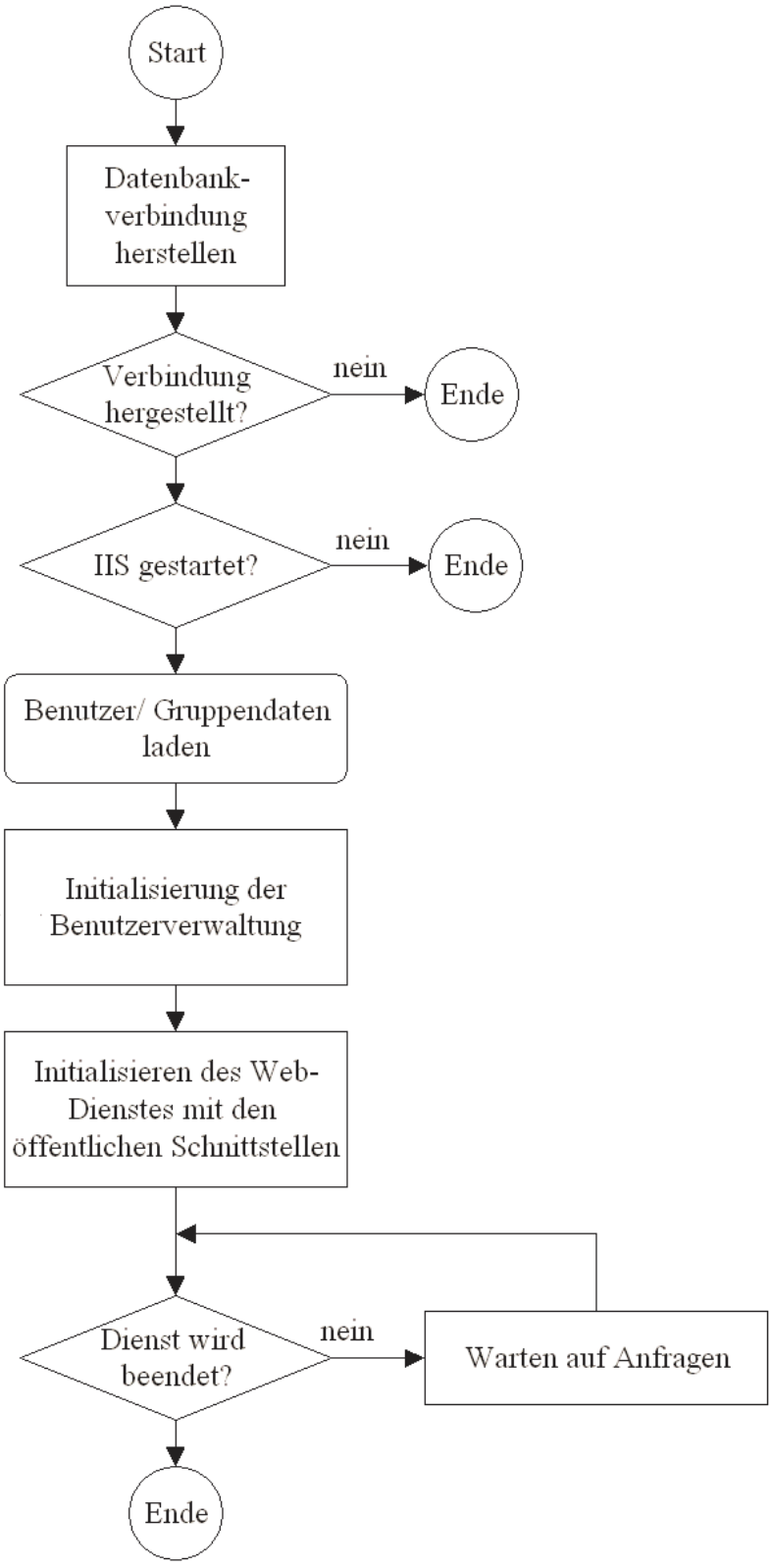


Abbildung 5.8: Start des SLM-Server-Dienstes (schematisch)

5.3.1 Zugriff auf die Datenbank

Die für den Datenbankzugriff verwendete Architektur ADO.NET bietet verschiedene Vorteile gegenüber den älteren Technologien OLE-DB und ADO. Zu diesen gehören:

- Verwendung des XML[21]-Datenaustauschstandards [23], der die Beschreibung jedes beliebigen Datenformats erlaubt und dadurch die Umwandlung der Datenformate in die des COM-Standards und zurück während des Datenaustauschs überflüssig macht.
- Minimierte offene Verbindungen. Alle Verbindungen zu einer Datenbank bestehen nur so lange, wie die jeweilige Datenbankoperation dauert, um die benötigten Ressourcen zu minimieren und die Zugriffe zu beschleunigen.
- Verbesserter Zugriff auf die Daten durch das generalisierte Dataset-Objekt, welches nicht nur einzelne Tabellen abzubilden vermag, wie das entsprechende Recordset-Objekt in ADO [23], sondern komplexe Tabellenstrukturen und deren Verknüpfungen untereinander.

Die Architektur von ADO.NET besteht im wesentlichen aus zwei Hauptkomponenten, dem .NET Datenprovider und dem DataSet-Objekt. Das DataSet wurde explizit für den Datenzugriff unabhängig von Datenquellen entworfen. Es bietet gegenüber anderen Zugriffsverfahren den Vorteil, dass es mit den unterschiedlichsten Datenquellen, wie beispielsweise XML-Daten, oder lokalen Anwendungsdaten verwendet werden kann. Das DataSet enthält eine Auflistung von einem oder mehreren DataTable-Objekten, die aus Datenzeilen und -spalten sowie aus Primärschlüsseln, Fremdschlüsseln, Einschränkungen und Beziehungsinformationen über die Daten in den DataTable-Objekten besteht [21].

Der .NET-Datenprovider beinhaltet die folgenden Objekte für den Datenzugriff auf die Datenbank: das Connection-Objekt, das Command-Objekt, das DataReader-Objekt und das DataAdapter-Objekt. Das Connection-Objekt (Verbindung) sorgt für eine Verbindung zur Datenquelle. Mit dem Command-Objekt wird auf Datenbankbefehle zugegriffen, um Daten

[21] Extensible Markup Language

zurückzugeben oder zu ändern und Parameterinformationen zu senden oder abzurufen. Das DataReader-Objekt sorgt für den Datenaustausch zwischen dem .NET-Dataprovider und der Datenquelle. Das DataAdapter-Objekt fungiert als Brücke zwischen dem DataSet-Objekt und der Datenquelle. Das DataAdapter-Objekt verwendet Command-Objekte zum Ausführen von SQL-Befehlen in der Datenquelle, um das DataSet mit Daten zu laden und Änderungen an den Daten im DataSet in die Datenquelle zu übernehmen.

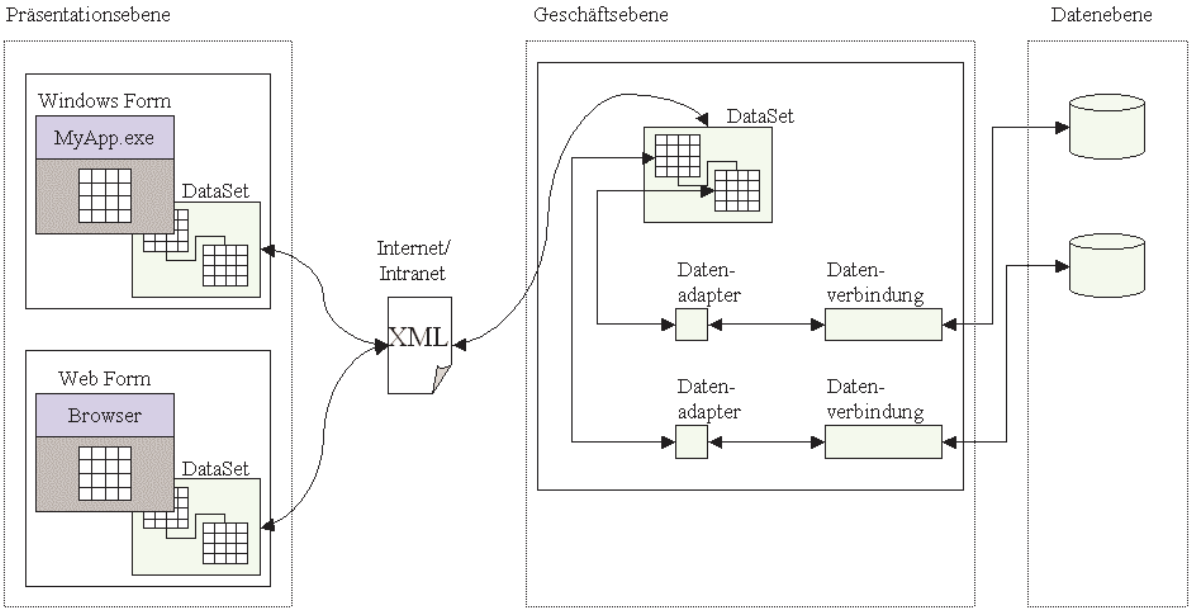


Abbildung 5.9: Hauptkomponenten des SLM-Systems (Datenzugriff und Datenaustausch)

Die Architektur des SLM-Systems, das unter Verwendung der ADO.NET-Technologie entwickelt wurde, besteht im wesentlichen aus einer 3-Ebenenarchitektur, die sich in Daten-, Geschäfts- und Präsentationsebene gliedert (vgl. Abbildung 5.9). Die unterste Ebene bildet die Datenebene, die für die eigentliche Datenhaltung verantwortlich ist. In der Geschäftsebene ist die Logik der tatsächlichen Datenverarbeitung untergebracht. Wie dargestellt, sorgen Datenverbindungsobjekte (Connection-Objekte) für den Zugriff auf die Datenebene, wobei für verschiedene Arten von Datenquellen spezielle Datenverbindungsobjekte existieren. Über den Datenadapter werden die aus den Datenquellen abgerufenen Daten in das DataSet-Objekt und seine zugehörigen DataTable-Objekte übertragen. Die Übertragung der Daten in die Präsentationsebene, welche im wesentlichen die Benutzersicht der Daten repräsentiert,

geschieht über das Inter/Intranet mittels des XML-Datenformats. Die Datenhaltung auf der Präsentationsebene geschieht ebenfalls in DataSet-Objekten (vgl. Abbildung 5.10).

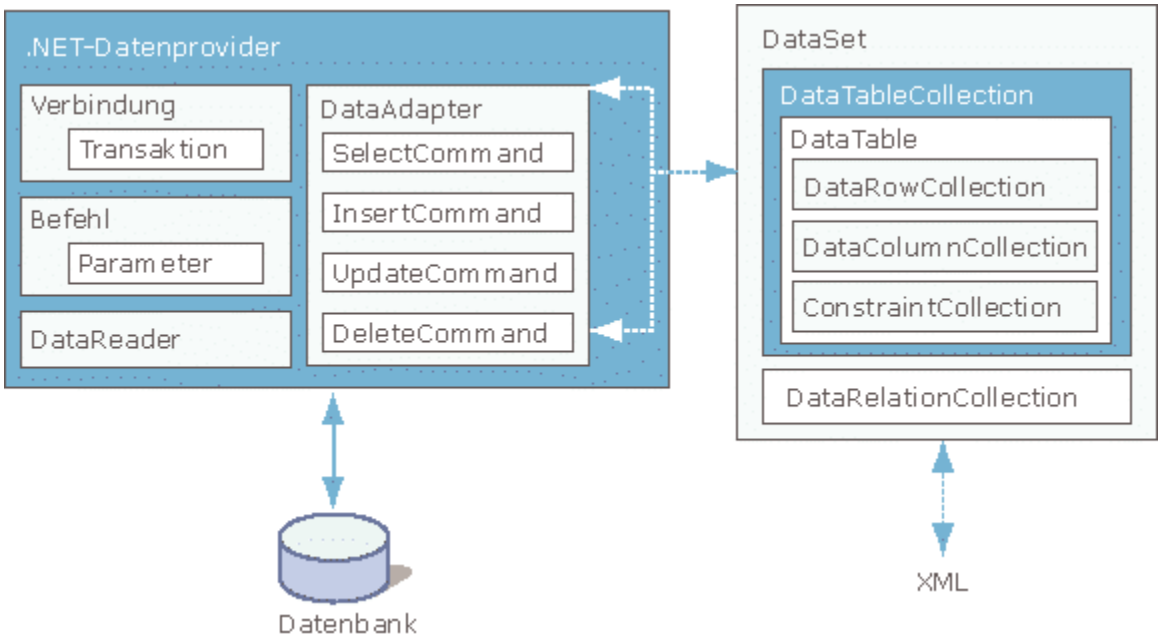


Abbildung 5.10: ADO .NET Architektur [24]

Der Aufbau einer Datenverbindung zu einer Datenquelle mit Hilfe von ADO.NET aus einem Browser mit Visual Basic.NET heraus, um dort eine Tabelle der Autoren zu erhalten, die in der Datenquelle abgelegt sind, sieht folgendermaßen aus (vgl. Listing 1):

Zunächst wird ein Connection-Objekt erzeugt, unter Angabe des Servers (hier der lokale Server; es kann aber auch ein Remote-Server angegeben werden), des Benutzers mit Kennwort und der gewünschten Datenquelle. Als nächstes wird ein SQL-Datenadapter erzeugt, mit Angabe der gewünschten Abfrage und der zu verwendenden Connetion. Über die „Fill“-Methode des Datenadapters wird nun die gesamte Verbindungs- und Abfragekaskade gestartet. Das Ergebnis der Abfrage wird in das zuvor erzeugte DataSet-Objekt gespeichert und bei der Rückgabe an den Aufrufer automatisch über XML serialisiert. Die erzeugten Objekte werden durch die .NET Laufzeitumgebung zerstört.

```
<WebMethod()> Public Function GetAuthors() As System.Data.DataSet

      ' Connection-Objekt erzeugen unter Angabe der Datenbank (pubs),
      ' des Datenbankservers (Lokaler Server) und des Benutzers (sa)
      Dim Connection As SqlConnection
      Connection = New _
            SqlConnection("server=(local);uid=sa;pwd='';
                        database=pubs")

      ' DataAdapter-Objekt erzeugen und mit Abfragekommando
      ' initialisieren
      Dim Adapter As SqlDataAdapter
      Adapter = New _
            SqlDataAdapter("select * from Authors",Connection)

      ' leeres DataSet-Objekt erzeugen
      Dim DS As Data.DataSet
      DS = New Data.DataSet()

      ' DataSet-Objekt mit Daten füllen
      Adapter.Fill(DS,"Authors")

      ' DataSet-Objekt an den Aufrufer zurückgeben

      Return DS
End Function
```

Listing 1: ADO-Datenverbindung mit VB .NET

Für den Aufbau der Verbindung zur Datenquelle wird ein fest vorgegebener Connection-String verwendet. Ebenso wird für das Abfragekommando eine fest vorgegebene Zeichenfolge eingesetzt. Mit der „*Return*"-Anweisung werden schließlich die angefragten Daten an den „*Aufrufer*" der Methode zurückgegeben.

Im folgenden werden nun die einzelnen Funktionalitäten des Servers und die damit verbundenen Strukturen der Datenbank vorgestellt.

5.3.2 Aufbau der Klassen

In der Datenbank des SLM-Systems existieren verschiedene Tabellen, auf die über ADO.NET zugegriffen werden muss. Hierbei gibt es spezialisierte Tabellen, die zu jeder Zeit in der Datenbank existieren und nicht in ihrem Aufbau verändert werden. Auf der anderen Seite existieren Tabellen, die durch die Benutzer konfiguriert werden können, deren Aufbau also bei der Programmerstellung nicht bekannt ist und auf die zur Laufzeit zugegriffen werden muss.

Hierbei existieren Zugriffe, die für alle Tabellen der Datenbank immer wieder durchzuführen sind, wie z.B. die Suche nach dem Primärschlüssel einer Tabelle. Eine der wichtigsten Aktionen, die auch immer wieder ausgeführt werden muss, ist der Aufbau der Verbindung zur Datenbank, wie Listing 1 zeigt.

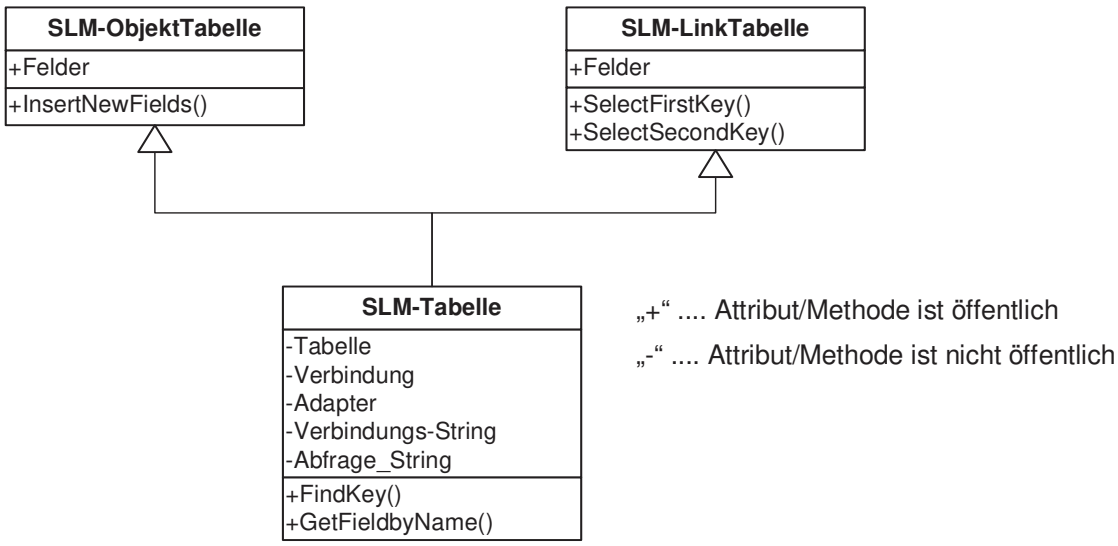


Abbildung 5.11: Klassenhierarchie für Tabellenzugriff (UML)

Dies kann zusammengefasst werden, indem eine allgemeingültige Klasse SLM-Tabelle (vgl. Abbildung 5.11) erzeugt wird, die für alle Datenbankzugriffe zuständig ist. Diese Klasse

besitzt als Attribute die notwendigen Objekte und Informationen, um die Datenbankverbindung aufzubauen und als Methode einige generelle Funktionalitäten, die für alle Tabellen der Datenbank gleich sind. Für spezielle Tabellen, insbesondere seien hier die Objekt-Tabelle genannt und die Link-Tabelle, werden spezialisierte Objekte zur weiteren Vereinfachung des Datenzugriffs abgeleitet. Als Attribute besitzen diese Objekte insbesondere sämtliche Felder eines Datensatzes, was den Zugriff darauf nochmals vereinfacht.

5.3.3 Lebenszyklus

Der Lifecycle, also der Lebenszyklus eines Objektes wird durch verschiedene Zustände und deren Übergänge beschrieben. Die Zustände, die ein Objekt zunächst sinnvollerweise annehmen kann, sind folgende:

- New: Das Objekt wird neu in der Datenbank angelegt. Zugehörige Dokumente werden automatisch erzeugt und notwendige Metadaten, wie z.B. Erstelldatum, Ersteller, Bezeichnung, usw. werden eingetragen. Die eigentlichen, dem Objekt zugeordneten Dokumente sind noch nicht hinterlegt.
- Checked In: Die eigentlichen Dokumente werden in den Vault übertragen und sind somit für alle sichtbar.
- Released: Die Dokumente sind für die weitere Verwendung freigegeben.
- Checked Out: Das Dokument wird von einer berechtigten Person bearbeitet und ist für die Bearbeitung von anderen gesperrt. Beim neuerlichen Check In wird eine neue Version des Dokumentes angelegt; die alte wird nicht überschrieben.
- Obsolete: Die Version des Dokumentes ist, es existiert eine neuere
- Disabled: Das Dokument darf nicht verwendet werden

Weitere sinnvolle Zustände, die ein Objekt annehmen kann, sind z.B. „*geprüft*“,“ *fehlerhaft*“ oder ähnliche, die aber gegebenenfalls in der Datenbank ergänzt werden können. Unabhängig von ihrem jeweiligen Status können sämtliche Dokumente jederzeit von berechtigten Personen eingesehen werden. Auch zusätzliche Zustände müssen nachträglich in das System

integriert werden können. Ein einfaches Freigabeschema für das SLM-System, welches als Standardkonfiguration und Vorlage für die Erweiterung dient, wird im SLM-System vorgegeben (vgl. Abbildung 5.12):

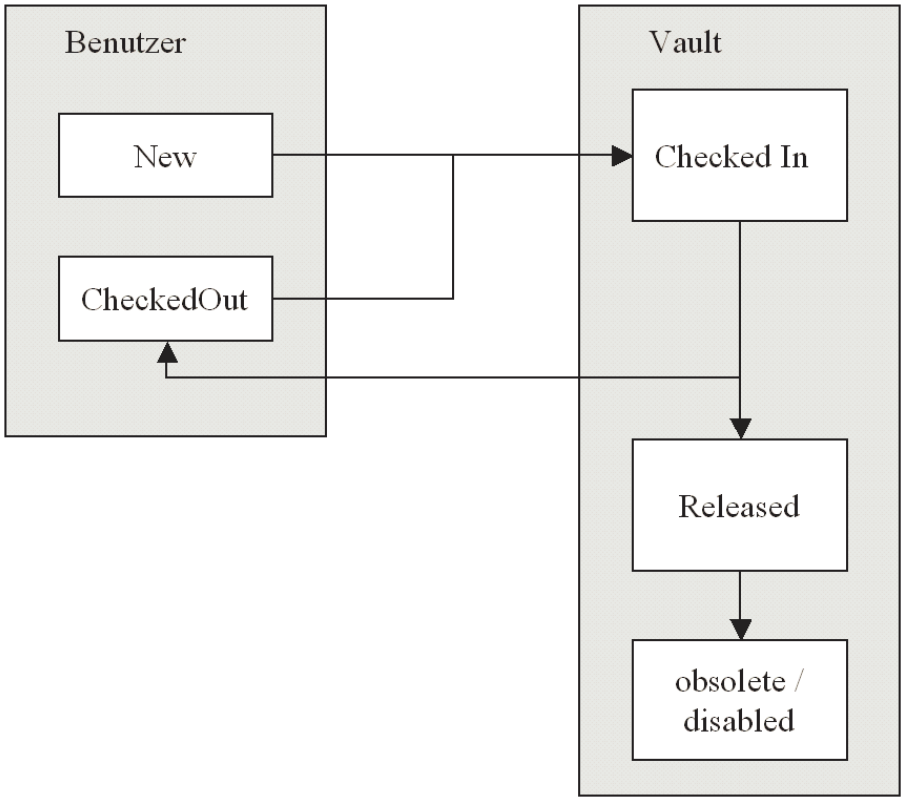


Abbildung 5.12: einfaches Freigabeschema

Dieses Freigabeschema kann entsprechend den Projekterfordernissen erweitert werden.
Die einzelnen Zustandsänderungen werden projektbezogen von verschiedenen Personen durchgeführt, deren Rolle im Projekt durch eine Gruppenzugehörigkeit charakterisiert werden kann. Hierbei existieren zwei verschieden Arten von Gruppen:

- „natürliche Gruppen“, die durch den Zusammenhang einer Person mit einem Objekt gegeben sind, also z.B. Ersteller / Besitzer eines Dokumentes.
- „künstliche Gruppen“, die von Dritten in einer Tabelle der Datenbank festgelegt werden, wie z.B. Entwickler, Projektleiter oder Tester.

Diese Zustände können nun von einzelnen Personen, die bestimmte Funktionen innerhalb eines Projektes wahrnehmen, geändert werden. Jedoch kann und darf nicht jede Person jede Änderung vornehmen und es ist auch nicht jeder Statusübergang möglich bzw. sinnvoll. Die Statusübergänge im SLM-System können nur auf zwei verschiedene Arten realisiert werden.

- durch feste Vorgabe der Software oder
- durch Festlegung in der Datenbank.

Die feste Vorgabe in der Software ist unflexibel und benötigt eine Anpassung der Software, wenn das vorgegebene Freigabeschema nicht den Erfordernissen entspricht. Die Festlegung des Freigabeschemas in der Datenbank ermöglicht eine wesentlich flexiblere Handhabung, benötigt aber bei der Einrichtung einen etwas erhöhten Aufwand, da jede Statusänderung für jede Funktionsgruppe genau festgelegt werden muss. Deswegen muss zur Erleichterung der Arbeit ein rudimentäres, vorkonfektioniertes Freigabeschema bereits vorhanden sein. Für die Durchführung von Zustandsänderungen durch das SLM-System gilt folgende Prinzip:

Jede Statusänderung, die nicht explizit erlaubt ist, ist verboten.

Es muss also für jede Personengruppe innerhalb eines Projekts festgelegt werden, welche Statusänderungen für diese im einzelnen erlaubt sind. Für bestimmte Personengruppen sind aber auch Verbote von Statusänderungen sinnvoll. Insbesondere dann, wenn einzelne Personen mehreren Gruppen angehören. Als Beispiel sei hier z.B. der Projektleiter genannt, der in dieser Funktion z.B. einzelne Dateien freigeben darf. Als Ersteller einer spezifischen Quelldatei sollte es auch für den Projektleiter nicht möglich sein, diese freizugeben, da er bei der Prüfung seiner eigenen Datei befangen ist, was dazu führen kann, dass er Fehler übersieht. Diese Prüfung und Freigabe sollte also immer von einer dritten Person durchgeführt werden.

Zustandsänderung			
Zustand alt	Zustand neu	Erlaubt	Gruppe
A	B	ja	Prüfer
A	B	nein	Besitzer

Abbildung 5.13: Zustandsänderung

In einer Tabelle werden alle Zustandsänderungen mit dem Hinweis erlaubt oder nicht erlaubt abgelegt (vgl. Abbildung 5.13). Dabei ist es völlig gleichgültig, um welche Zustandsänderung es sich handelt. Soll nun durch eine bestimmte Person eine Zustandsänderung durchgeführt werden, so wird zunächst geprüft, ob für alle Gruppen, denen diese Person angehört, diese Zustandsänderung verboten ist. In einem zweiten Schritt wird geprüft, ob diese für mindestens eine der Gruppen erlaubt ist, der diese Person angehört. Erst danach wird diese Zustandsänderung durchgeführt.

Zunächst wird geprüft, ob die Person, die den Zustand des Objektes X von A nach B ändern möchte, einer Gruppe angehört, für die diese Zustandsänderung verboten ist. Sollte dies nicht der Fall sein, so wird geprüft, ob für eine der Gruppen, denen diese Person angehört, diese Zustandsänderung erlaubt ist. Gehört also die Person in obigem Beispiel der Gruppe „Prüfer“ an, so ist die Zustandsänderung A->B für diese Person erlaubt, sofern sie nicht auch der Gruppe „Besitzer“ angehört.

5.3.4 Anlage von Metadokumenten

Das Arbeiten mit dem SLM-System führt permanent zur Anlage neuer Objekte in der Datenbank, die von der Person, die das Objekt anlegt, in geeigneter Weise dokumentiert werden muss. Aus Gründen der Konsistenz sollte dies am besten sofort bei der Erstellung des neuen Dokumentes geschehen. Der häufigste Fall einer Neuanlage von Dokumenten ist sicherlich das Aus- und Einchecken einer Quellcodedatei. Beim neuerlichen Einchecken der Quellcodedatei kann es zu einer Änderung der Vorgängerversion gekommen sein, die aus Gründen der Übersichtlichkeit in einem Änderungsprotokoll mitverfolgt werden muss.

In ähnlicher Weise müssen für verschiedene Arten von Objekten und Dokumenten in der Datenbank sofort bei der Neuerstellung Metadokumente angelegt werden, um einer etwaigen, nachträglichen, ungenauen oder gar nicht erfolgten Dokumentation vorzubeugen. Beispiele hierfür sind das Pflichtenheft oder die Funktionsliste, die beim Anlegen eines neuen Projektes generiert werden, oder ein UML-Dokument bei der Anlage einer neuen Quellcode-Datei.

Die verschiedenen im SLM-System anzulegenden Dokumenttypen können anhand ihres Dokumenttyps und ihrer Projektzugehörigkeit unterschieden werden. So gehören zunächst alle Dokumente zum Typ Objekt in der Datenbank und werden mittels Verweisen auf weitere Tabellen diversifiziert, in denen noch weitere Verweise auf Untertabellen vorhanden sein können. Wird nun z.B. eine neue ME10-Makrodatei in die Datenbank eingefügt, so hat diese Datei zunächst einmal einen Projektbezug und ist über die Typklassifizierung als ME10-Makrodatei gekennzeichnet. Anschließend werden der Reihe nach die Metadokumente erzeugt und dem spezifischen Entwickler vorgelegt, der diese ME10-Makrodatei erzeugt hat und die in dem Projekt für den Dokumenttyp ME10-Macrodatei konfiguriert wurden.

Da es aufwändig ist, für jedes Projekt diese Definition neu vorzunehmen, ist es zweckmäßig, eine Standardkonfiguration vorzugeben, die automatisch für jedes neu angelegte Projekt übernommen wird und dann den Erfordernissen angepasst werden kann.

Tabelle Metadokumente

ID	Project	DocumentType	ObjectID
1	0	12	98
2	0	23	87
3	123	12	98
4	123	23	87
5	123	34	76
6	234	12	98
7	234	23	87

Tabelle Dokumenttypen

ID	Bezeichnung
12	Textdatei
23	ME10-Macro
34	C++Quelldatei

Abbildung 5.14: Tabellenstruktur zur Abbildung der Funktionalität

In der Tabelle, in der die Anlage neuer Metadokumente spezifiziert wird, existieren die Projekte mit den IDs „0“, „123“ und „234“ (vgl. Abbildung 5.14). Die Projekt-ID „0“, die keinem speziellen Projekt zugeordnet ist, denn das Projekt mit der ID „0“ existiert in der Datenbank nicht, fungiert hierbei als Vorlage für die Anlage von Metadokumenten in neuen Projekten. Wird also ein neues Projekt angelegt, wie in diesem Fall die Projekte „123“ und „234“, so werden zunächst die Zuordnungen für die Anlage von Metadokumenten für diese Projekte übernommen, wie unter „0“ vorgegeben. Diese können vom Administrator bzw. vom Projektleiter um weitere Zuordnungen ergänzt werden, wie in obigem Beispiel unter der ID „5“. Wird nun im Kontext des Projektes „123“ ein neues Dokument vom Typ „34“, also eine C++-Quelldatei angelegt, wie aus der Tabelle für die Dokumenttypen hervorgeht, so wird automatisch ein neues Dokument vom Objekttyp „76“ angelegt, dem Ersteller der C++-Quelldatei vorgelegt und nach dem einchecken über einen Dokumentationslink mit der C++-Quelldatei verknüpft.

5.3.5 Workflow

Das zentrale Element eines Workflows ist ein Prozess. Ein Prozess ist eine definierte Folge von Arbeitsabläufen, die mit bestimmten Personengruppen, bzw. Personen verbunden sind [25]. Der Workflow sorgt für die automatische Weiterleitung von Daten und Informationen an die richtigen Stellen in dem jeweiligen Projekt (vgl. Abbildung 3.2). Ein solcher Workflow-Prozess ist durch die Weitergabe an eine Person/Personengruppe und eine damit verbundene

Statusänderung der mit dem Prozess verknüpften Dokumente gekennzeichnet. Hierbei wird jeweils immer diejenige Statusänderung durchgeführt, die im Prozess definiert ist. Die Benachrichtigung der betroffenen Person oder Personengruppe, die als nächstes für die Bearbeitung des Prozesses vornehmen sollen, erfolgt, sobald diese Person im SLM-System angemeldet ist. Ist die Bearbeitung nicht einer spezifischen Person, sondern einer Gruppe zugeordnet, so werden alle Personen benachrichtigt, die dieser Gruppe angehören. Eine beliebige Person dieser Gruppe kann dann den Prozess übernehmen, die damit hierarchisch verknüpften Dokumente bearbeiten und die vorgesehene Statusänderung durchführen oder gegebenenfalls den Prozess abweisen und damit zur erneuten Bearbeitung an die vorhergehende Instanz zurückgeben. Während des Prozessdurchlaufes sind die mit ihm hierarchisch verlinkten Objekte für andere gesperrt

Die Realisierung eines solchen Prozesses geschieht über einen speziellen Link-Typ, dem Prozess-Link, der über die Prozess-ID dem jeweiligen Prozess zugeordnet ist, über den in einer Kette abwechselnd Personen/Gruppen und die jeweiligen Statusänderungen miteinander verknüpft werden. Ein weiterer spezieller Link, der Status-Link, verbindet den konkreten Prozess mit der gerade bearbeitenden Person und zeigt damit an, an welcher Stelle der Prozessdurchlauf momentan steht. Einem solchen Prozess können nun auf zweierlei Arten beliebig viele Objekte zugeordnet werden:

- Über hierarchische Links: Die jeweiligen Dokumente unterliegen den Statusänderungen.
- Über einfache Links: Die Dokumente unterliegen nicht den Statusänderungen, sondern dienen nur der Information der jeweiligen Person.

Hierbei spielt es keine Rolle, ob es sich bei den zugeordneten Dokumenten um physikalische Dateien handelt, die im Vault des SLM-Systems vorhanden sind oder um abstrakte Objekte, die nur in der Datenbank abgebildet werden, wie etwa ein Projekt.

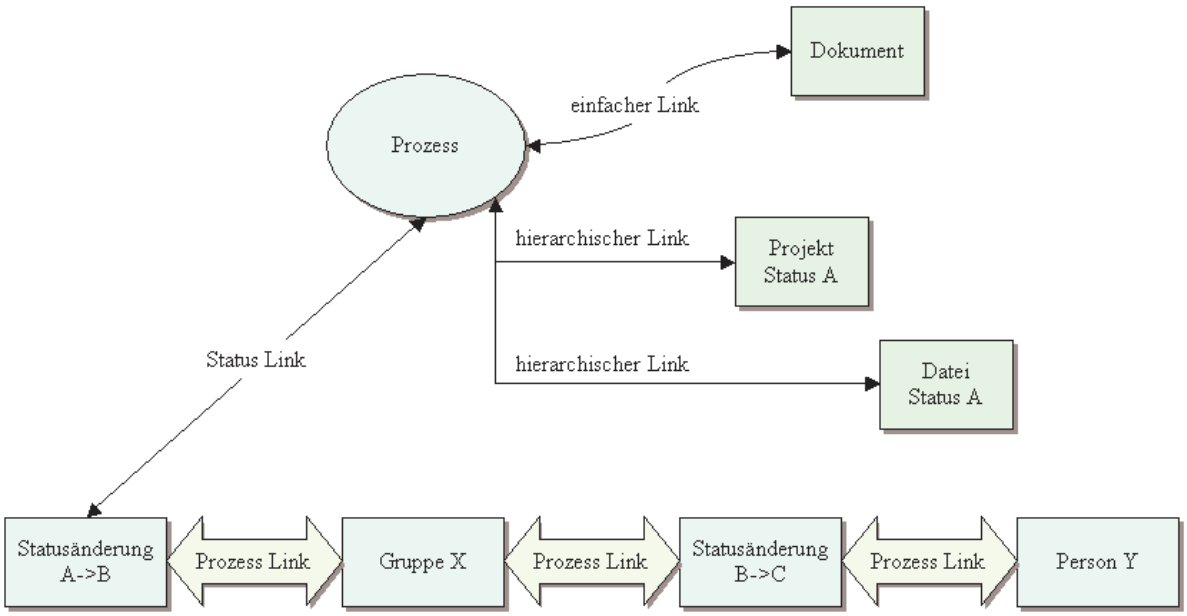


Abbildung 5.15: Prozessdefinition (Schematisch) vor Prozessstart

Im folgenden wird der prinzipielle Ablauf verdeutlicht. Es existiert ein Prozess (vgl. Abbildung 5.15), dem drei Objekte, zwei davon hierarchisch, zugeordnet wurden. Diesem Prozess ist des weiteren eine Prozesskette zugeordnet, die sich über verschiedene Personen und Statusänderungen erstreckt. Dabei existieren diese Personen und Statusänderungen unabhängig von den jeweiligen Prozessen und werden nur durch die jeweiligen Prozess-Links miteinander verknüpft, die über ein Attribut dem Prozess zugeordnet sind.

Nach Durchlauf des gesamten Prozesses (vgl. Abbildung 5.16) wurden an den dem Prozess hierarchisch zugeordneten Objekten die notwendigen im Prozess definierten Statusänderungen vorgenommen und abschließend die Person benachrichtigt, die als Prozessabschluss definiert wurde (in diesem Fall Person Y). Zu Dokumentationszwecken muss festgehalten werden, wer wann welchen Prozessschritt durchgeführt hat. Hierzu wird im Prozessdurchlauf die Gruppe durch die konkrete Person ausgetauscht, die den Prozessschritt bearbeitet hat, in diesem Fall durch die Person Z, damit nachvollziehbar wird, welche Person aus der Gruppe X den Prozess bearbeitet hat. Der Zeitpunkt der Bearbeitung wird im jeweiligen Prozess-Link bei dessen Durchlauf festgehalten.

Beispiele für solche Prozesse sind:

- Ein Entwickler checkt eine bestimmte Quellcodedatei und die zugehörige Header-Datei in das SLM-System ein, instantiiert einen neuen Prüfprozess, an den er diese hierarchisch linkt, ebenso wie eine Informationsdatei über die Funktionsweise des Quelltextes und startet den Prozess. Der Prozess weiß über den Status Link, welches der erste auszuführende Prozessschritt ist, gibt in diesem Fall den Quelltext zur Prüfung frei. Über die Prozess-Identifikationsnummer werden in der Datenbank die dem Prozess zugehörigen Prozess-Links identifiziert, mit der die einzelnen Prozessschritte miteinander verlinkt sind. Über diese wird der nächste Prozess-Schritt identifiziert und der Status-Link des Prozesses darauf gesetzt. Automatisch wird die Information über den Prozess nun an die Qualitätskontrolle weitergegeben. Dort übernimmt eine Person aus dieser Gruppe den Prozess, prüft und testet den Quelltext und die Header-Datei unter Berücksichtigung des beigefügten Informations-dokumentes und weist danach entweder den Prozess zurück zum Entwickler, wobei er ein weiteres Dokument mit den Beanstandungen anhängen kann, oder gibt den Prozess wieder frei. Wird der Prozess freigegeben, so wird zu Dokumentationszwecken zunächst die Gruppe durch die spezifische Person aus dieser ersetzt, welche die Bearbeitung tatsächlich vorgenommen hart, und es werden dann in der Prozesskette über die Prozess-Links, die nächsten Arbeitsschritte identifiziert und in diesem Falle die Quelltext und die Header-Datei mit dem Status „*Tested*“ versehen. Danach wird der Projektleiter über die Prüfung der Datei informiert, der diese entweder zurückweisen kann, oder gegebenenfalls die Datei nach seinem Ermessen freigeben kann.
- Der Vertrieb erhält den Auftrag eines Kunden für ein Softwareprojekt und legt ein neues, leeres Projekt im SLM-System an. Es wird ein Projektleiter bestimmt und dieser erhält automatisch die notwendigen Informationen über das neue Projekt, zu denen z.B. die Projektspezifikation gehört. Der Ablauf gestaltet sich ähnlich zu dem vorher beschriebenen Vorgang.

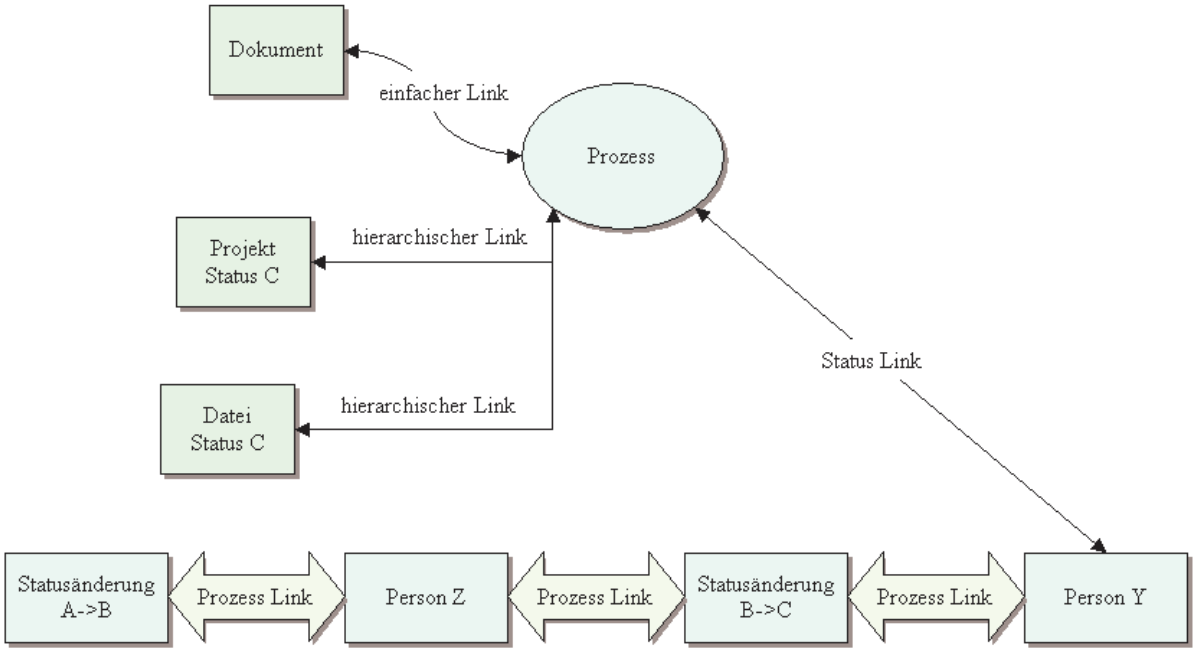


Abbildung 5.16: Prozess (schematisch) nach Durchlauf

Die Konfiguration der Prozessvorlagen erfolgt in der Datenbank in derselben Struktur, wie der eigentliche später ablaufende Prozess, bis auf den Umstand, dass dem Prozess zunächst keine weiteren Objekte zugelinkt sind. Hierbei wird der Prozess über ein Attribut als Prototypprozess gekennzeichnet. Bei der Konfiguration eines Prozesses durch eine berechtigte Person wird gleichzeitig vom SLM-System geprüft, ob die vorgesehenen Statusänderungen durch die definierten Personen bzw. Gruppen erlaubt sind, da sonst ein Prozessdurchlauf nicht möglich ist. Wird nun ein konkreter Prozess durch eine Person initiiert, so wird eine neue Instanz dieses Prozesses aus der Vorlage erzeugt, ebenso wie alle verwendeten Links. Diesem konkreten Prozess können nun Dokumente zugelinkt werden. Über die Metadaten der einzelnen Prozessschritte wird festgehalten, wer wann welche Bearbeitung vorgenommen hat.

5.3.6 Installationsversionen

Zur Auslieferung der Software an den Kunden wird diese in sogenannte Installationsprogramme „*verpackt*“. Diese Installationsprogramme sind in der Regel ausführbare Dateien, die alle Informationen der zu installierenden Software enthalten [26]. Sie werden vom

Kunden gestartet und kopieren die Software in geeigneter Weise auf dessen Rechnersystem. Bei dem Installationsvorgang muss eine Vielzahl von Randbedingungen berücksichtigt werden. So ist beispielsweise zu berücksichtigen, dass konfigurationsabhängig unterschiedliche Dateien auf das Zielsystem übertragen werden müssen. Ebenso erfordert die Installation eines Computerprogramms Manipulationen am Betriebssystem, wie beispielsweise das Setzen oder Verändern von Betriebssystemvariablen oder der Registrierungsdatenbank. Die Installation eines Computerprogramms stellt also in der Regel selbst einen komplexen Vorgang dar, der durch ein eigenes Programm bearbeitet wird. Aus diesem Grund verwendet man für die Erstellung von Installationsprogrammen spezielle Entwicklungsumgebungen, welche eine automatische Erstellung von Installationsmedien gestatten. Das Programm zur Erzeugung der Installationsversion benötigt die zu installierenden Dateien in bestimmten Verzeichnissen, entsprechend der vorgenommenen Konfiguration.

Die Verwaltung der Installationsversionen kann nun durch Installationslinks geschehen, mit denen die fertigen, zu installierenden Dateien mit der Installationsversion und der benötigten Verzeichnisstruktur verknüpft werden (vgl. Abbildung 5.17). Für den speziellen Typ des „*Installationslinks*“ existiert in der Datenbank eine weitere Tabelle, in der die Zuordnung der im Vault abgelegten Dateien mit der für das Werkzeug zur Generierung der Installationsversionen benötigten Namen und Dateien vorgenommen wird (vgl. Abbildung 5.18). Die ID in der Zuordnungstabelle entspricht dabei der ID des jeweiligen Installationslinks. Über die IDs (z.B. „12“) in der Zuordnungstabelle werden die im Vault vorhandenen Dateien mit dem Installationspfad (z.B. „/Setup/ProjX“) und dem Installationsnamen (z.B. „ProjX.exe“) verknüpft. Die Konfigurationsdatei für das Erstellungsprogramm kann durch einen einfachen Link mit dem entsprechenden Projekt verknüpft werden.

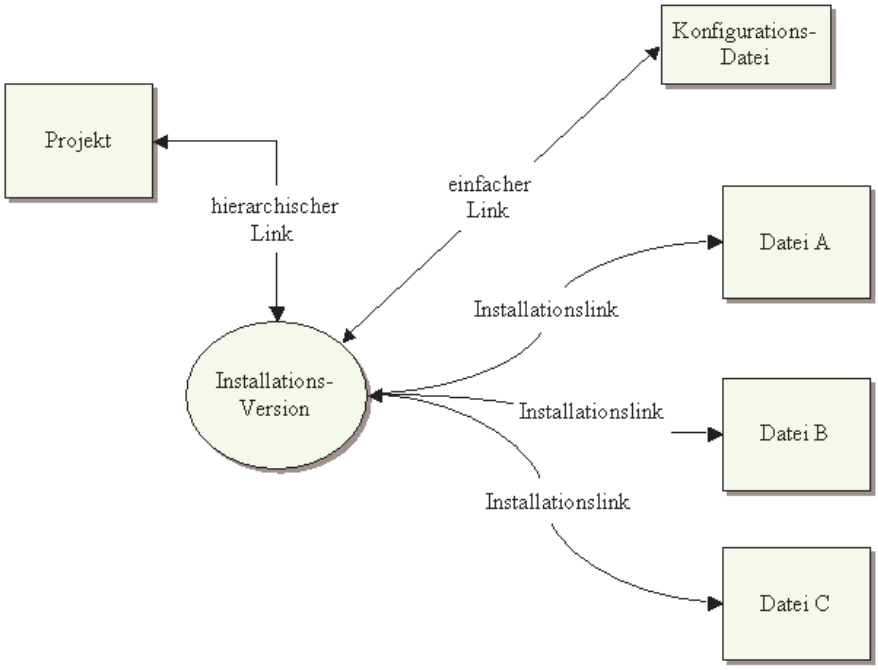


Abbildung 5.17: Konfiguration einer Installationsversion (schematisch)

Wird nun durch den Anwender des SLM-Sytems die Generierung einer Installationsversion angestoßen, so kann über die Versionshistorie der Dateien zunächst festgestellt werden, ob neue Versionen der Dateien vorhanden sind. Ist dies der Fall, so muss eine neue Version der Installationsversion erstellt werden, in der die neuen Dateiversionen der neuen Installationsversion zugeordnet werden, denn eine einmal freigegebene Installationsversion darf nicht geändert werden.

Die für die Installation notwendigen Dateien werden nun gemäß der Zuordnungstabelle über den Umweg des Austauschverzeichnisses unter dem Verzeichnis und dem Dateinamen auf den Client-Rechner kopiert, ebenso die Konfigurationsdatei für das Erstellungsprogramm der Installationsversion. Der Anwender kann nun auf dem Client das Installationserstellungs-programm starten und erhält automatisch die Installationsversion für das Software-Produkt. Gegebenenfalls kann auch das Installationserstellungsprogramm im SLM-System abgelegt werden, so dass es, falls notwendig, auch sofort auf dem Client-Rechner installiert werden kann.

Zuordnungstabelle

ID	Verzeichnis	Name
12	/Setup/ProjX	ProjX.exe
23	/Setup/ProjX	ProjX.ini
34	/Setup/ProjZ	ProjZ.exe

Abbildung 5.18: Zuordnungstabelle für Installationslink (beispielhaft)

5.3.7 Benutzerverwaltung und Zugriffssteuerung

Eine zentrale Anforderung an das SLM ist die Unterstützung von mehreren Benutzern, die gleichzeitig an einem oder mehreren Projekten arbeiten. Dabei haben unterschiedliche Personen verschiedene Funktionen innerhalb des gesamten Bearbeitungsspektrums. Um die einzelnen Benutzer voneinander unterscheiden zu können, müssen diese vom SLM-System verwaltet werden. Zu den Informationen, die verwaltet werden müssen, gehören

- der Name,
- der Ort, an dem sich die Person gewöhnlich aufhält (z.B. Raumnummer des Büro),
- Informationen über die Erreichbarkeit (z.B. Telefonnummer, E-Mail Adresse),
- der Login-Name, unter der sich diese Person am SLM-System anmeldet und
- das zugehörige, persönliche, verschlüsselte Kennwort.

Da die verschiedenen Personen auch Teile von Prozessen sein können (vgl. 5.3.5) und daher „verlinkbar“ sein müssen, können sie nicht unabhängig von den anderen Objekten in der Datenbank verwaltet werden, sondern müssen Teil der Objekthierarchie sein, also vom Typ Objekt sein.

Jede Person, die mit dem SLM-System arbeitet, hat in einem Projekt bestimmte Aufgaben zu erledigen, die durch besondere Rechte und Pflichten gekennzeichnet sind. Die Aufgaben und Rechte einer bestimmten Person können von Projekt zu Projekt unterschiedlich sein. Es ist daher erforderlich, die Personen projektbezogen zu Gruppen zusammenzufassen um die Zugriffe und Statusänderungen zu steuern. Diese Gruppen entscheiden darüber, welche

Rechte, also welche Möglichkeiten zur Statusänderung (vgl. 5.3.3) eine Person generell in einem Projekt besitzt. Zusätzlich zu diesen Projekt bezogenen Gruppen existieren auch noch Gruppen, die in allen Projekten bestimmte Rechte besitzen. Die sicherlich wichtigste dieser Gruppen ist die Gruppe „Administratoren“, die über prinzipiell alle Rechte in allen Projekten verfügt und auch Änderungen an der Struktur der Datenbank vornehmen darf.

Für die Authentifizierung der Benutzer wird die Kombination aus Login-Name und dem zugehörigen Kennwort verwendet. Aus Sicherheitsgründen müssen diese Kennwörter verschlüsselt werden und ebenso die Übertragung der Authentifikationsinformationen zwischen Client und Server. Diese Aufgabe übernimmt das .NET-Framework über die so genannte Passport-Authentifikation. Die Datenübertragung erfolgt dabei über den SSL (Secure Socket Layer) von Windows, der die Triple-DES[22] Verschlüsselung verwendet. Der Client sendet eine Anforderung an den Server und erhält eine Anmeldeanforderung zurück. Der Benutzer gibt seine Anmeldeinformationen über einen Dialog ein und der Client erhält ein sogenanntes Ticket zurück, welches bei weiteren Anfragen zur Authentifizierung dient. Serverseitig erfolgt die Identifikation der Benutzer über sogenannte Principal-Objekte, mit denen die Gruppenzugehörigkeiten der Benutzer über PassportIdentity-Objekte identifiziert werden. Die Informationen der Gruppenzugehörigkeiten der einzelnen Benutzer in den Principal-Objekten werden aus der Datenbank generiert [21].

Bei Aufruf der öffentlichen Funktionen des SLM-Systems wird durch die Übergabe des Ticket bei dieser Anforderung dann jeweils durch das .NET-Framework überprüft, ob der Aufruf rechtmäßig ist. Letztendlich sind dadurch alle Anfragen an das SLM-System auch durch die Triple-DES Verschlüsselung hinreichend sicher. Dies betrifft jedoch nicht den Datenaustausch zwischen Client und Server, der über XML und die Freigaben abläuft. Die Verschlüsselung dieser Daten würde zu einer problematischen Zunahme der Datenmengen führen und den Datenaustausch erheblich verlangsamen.

Das SLM-System ist als Mehrbenutzersystem konzipiert, da in der Regel mehrere Benutzer gleichzeitig mit dem System arbeiten. Die Benutzer melden sich dabei über den Client an

[22] Data Encryption Standard (Triple-DES - es wird eine Dreifachverschlüsselung verwendet)

(vgl. Abbildung 5.19) und ab, der jeweils lokal auf ihren Arbeitsplatzrechnern läuft. Bei der Anmeldung an den Server erhält der Client jeweils als Rückgabeparameter ein so genanntes Ticket (s.o.), welches jeweils die Sitzung des Benutzers im Server repräsentiert. Dieses Ticket wird bei jedem Aufruf des Clients einer der öffentlichen Schnittstellen des Servers mit übergeben und dient zur Identifikation des Clients und damit des angemeldeten Benutzers.

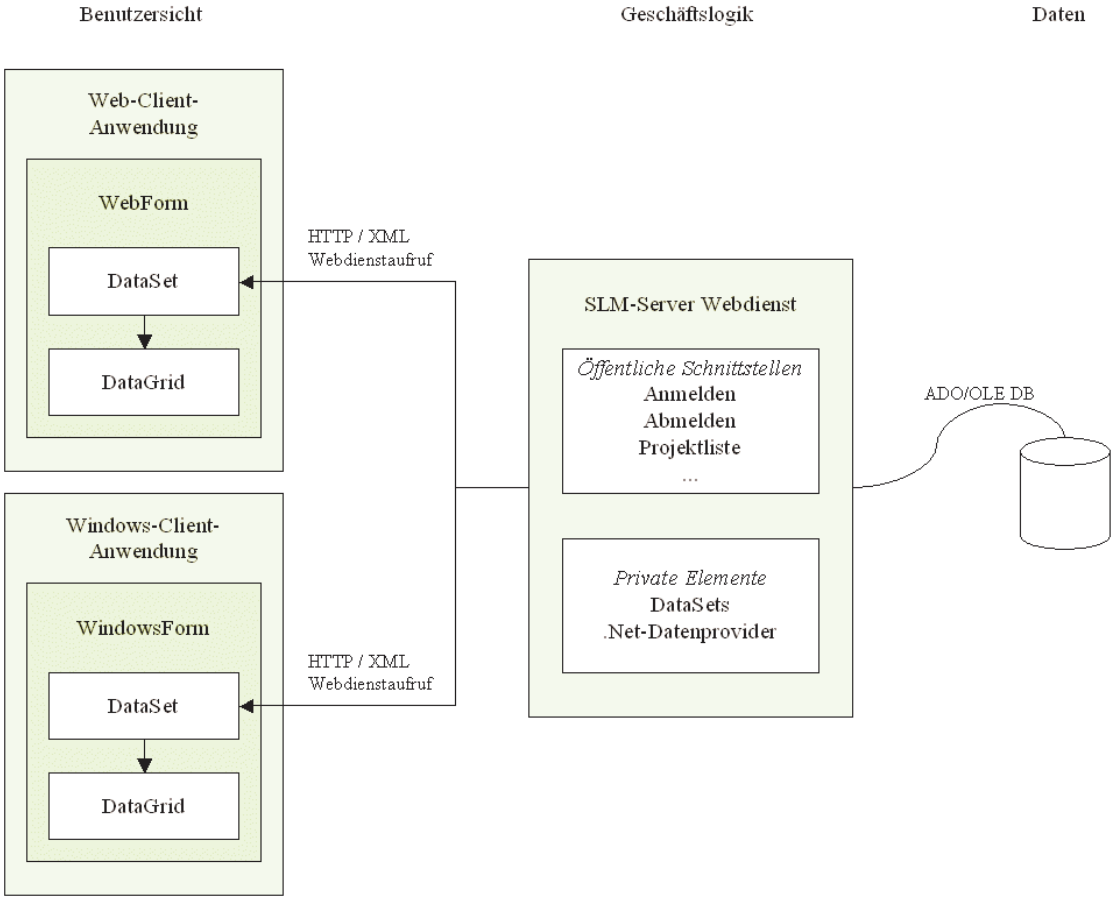


Abbildung 5.19: Anmeldung von Clients

Die Tickets werden auf Anfrage durch den Server generiert und mittels Principal-Objekten durch das .NET-Framework verwaltet. Bei der Abmeldung des Clients verliert das entsprechende Ticket seine Gültigkeit, so dass Anfragen mit diesem nicht mehr möglich sind.

Die Verwaltung von Benutzersitzungen geschieht über das .NET Framework. Im Prinzip sind alle Zugriffe auf das SLM-System öffentlich, jedoch wird der Zugriff darauf über die Passport-Authentifizierung des .NET Frameworks gesteuert. Die Zugriffe eines Clients sind

über die gesamte Gültigkeitsdauer des durch die Passport-Authentifizierung ausgestellten Tickets zulässig. Alle anderen Anforderungen werden abgelehnt. Eine explizite Verwaltung von Benutzersitzungen, etwa durch Session-Objekte durch das SLM-System existiert nicht.

5.3.8 Bereitstellung von Dateien

Die Hauptaufgabe des SLM-Systems ist die Verwaltung der im Softwareentwicklungsprozess entstehenden Daten, die zum großen Teil in Form von Dateien vorliegen. Diese Dateien werden serverseitig in einem Vault gespeichert. Hierbei wandelt das SLM-System den Dateinamen ab, da es sonst zur Namensgleichheit verschiedener Dateien aus verschiedenen Projekten kommen kann. Der Datenaustausch geschieht über eine gesonderte Freigabe. Werden von einem Client eine oder mehrere Dateien angefordert, werden diese aus dem Vault in die Freigaben hineinkopiert. Die ursprünglichen Dateien im Vault bleiben unverändert erhalten. Aus der Freigabe werden diese Dateien vom Client in das Zielverzeichnis verschoben. Eine genaue Beschreibung dieses Vorganges befindet sich in Kapitel 5.1.3.

5.3.9 Scriptsprache

Zur Anpassung des SLM-Systems an kundenspezifische Gegebenheiten, muss das System in der Lage sein, diese Anpassungen ohne Änderung des SLM-Systems durchzuführen. Ebenfalls muss für die Integration der einzelnen Komponenten eine Softwareentwicklungs-umgebung gegebenenfalls Scripte ausführen können. Die Ausführung der Scripte muss sowohl clientseitig als auch serverseitig möglich sein, um diese kundenspezifischen Anpassungen vornehmen zu können. Als API können die Zugriffsfunktionen des Clients auf den Server dienen. Da das SLM-System in Visual-Basic entwickelt wird, kann dieses als Scriptsprache problemlos zum Einsatz kommen. Die einzelnen Scripte können in der Datenbank und im Vault abgelegt werden und dann bei Bedarf entweder direkt vom Server oder auf dem Client ausgeführt werden. Die Scripte dienen dabei verschiedenen Aufgaben.

Clientseitig sind hier Aufgaben zu nennen, die der Kommunikation des Clients mit den einzelnen Programmen dienen, die mit dem Client integriert werden sollen. Serverseitig kann

hier die Funktionalität des SLM-Systems erweitert werden, denn ein auf dem Server ausgeführtes VB-Script kann auf internen Funktionalitäten des SLM-Systems zurückgreifen, hier insbesondere auf die Funktionalitäten der Werkzeugklasse und des Datenzugriffs, wobei das Script aber immer nur mit den Privilegien des über den Client angemeldeten Benutzers ausgeführt wird.

5.3.10 Klassifizierung

Die Klassifizierung der Daten ermöglicht die Suche nach bestimmten Dokumenten innerhalb des SLM-Systems. Für diese Klassifizierung können in erster Linie die Metadaten verwendet werden, die über die einzelnen Objekte gespeichert wurden. Mit Hilfe der Klassifizierung können gezielt bestimmte Dokumente oder Quellcodedateien für die Wiederverwendung gefunden werden. Die wichtigsten Klassifizierungs-Metadaten sind:

- Projektzugehörigkeit
- Dateityp
- Datum
- Ersteller
- Version

Im Hinblick auf das Ziel, der Wiederverwendung von Software, ist dies nicht ausreichend. Allerdings ist es äußerst schwierig, ein Klassifizierungsschema für bereits erstellte Software zu entwickeln, die es gestattet, ganz bestimmte erstellte Softwarekomponenten zu finden, um diese dann der Wiederverwendung zuzuführen und sie an die aktuellen Erfordernisse anzupassen. Daher ist es sinnvoll, bestimmte Dokumentklassen gezielt nach Stichworten zu durchsuchen, die für die gesuchte Software charakteristisch sind. Dies ist deshalb möglich, da im Konzept des SLM-Systems vorgesehen ist, dass zu allen im System vorkommenden Quellcodedateien entsprechende zugehörige Metadokumente existieren. Der Benutzer kann diese Metadokumente einsehen und er kann dann über die Verlinkung zwischen Metadokument und Quellcodedatei bei Übereinstimmung sofort auf die Quellcodedatei zugreifen.

5.3.11 Kommunikation mit dem Client

Die Kommunikation zwischen Client und Server geschieht über Web-Services, die über einen „Internet-Information-Server“[23] bereitgestellt werden. Jedoch ist auch zwischen Server und Client eine Kommunikation notwendig, insbesondere zur Ausführung von Aufgaben des Workflows. Aufgrund der Schwierigkeiten, die entstehen können, wenn ein Remote-Prozeduraufruf über ein Netzwerk ausgeführt werden soll, da diese im ungünstigsten Fall durch eine Firewall abgeblockt werden können, führt der Client in regelmäßigen Abständen Abfragen an den Server aus, ähnlich einem Mail-Client, um festzustellen, ob im Rahmen eines Workflow-Tasks Dateien zu bearbeiten sind. Soll eine direkte Kommunikation zwischen Server und Client stattfinden, um den Client vom Server aus Remote zu steuern, ist es notwendig, den Client ähnlich dem SLM-Server als Webdienst zu gestalten, was wiederum den Betrieb eines Internet-Information-Servers auf dem Client-Rechner und einen deutlich erhöhten Verwaltungsaufwand erfordert.

Die öffentlichen Schnittstellen sind Funktionen, die als Web-Service prinzipiell von jedem Benutzer aufgerufen werden können. Über die Passport-Authentifikation wird jedoch der Zugriff auf diese durch das .NET-Framework eingeschränkt. Sie beinhalten alle Funktionalitäten des SLM-Systems, auf die durch den Client zugegriffen werden soll. Die Schnittstellen greifen dabei auf die interne Geschäftslogik des SLM-Systems zu, welche durch die Werkzeugklasse (vgl. Abbildung 5.20) repräsentiert wird. Zu den wichtigsten dieser öffentliche Schnittstellen gehören:

- ClientAnmelden()
- ClientAbmelden()
- Workpoll() (routinemäßige Abfrage auf Nachrichten für den Workflow)
- GetProjectList()
- GetFileList()
- GetFileTypes()

[23] kostenlose Microsoft Betriebssystem Erweiterung zur Bereitstellung von Internetdiensten

- GetLinkedDocuments()
- ViewProject() (lädt Projekt zur Ansicht auf den Client, ohne auszuchecken)
- CheckOutProject()
- CheckInProject()
- ViewFile() (lädt Datei zur Ansicht auf den Client, ohne auszuchecken)
- CheckOutDatei()
- CheckInDatei()
- RunScript() (führt VB-Script auf Server aus, mit den Rechten des Clients)
- LinkFile() (Verlinkt zwei Dateien miteinander)

Die hier gezeigten öffentlichen Schnittstellen stellen nur eine kleine Menge der Gesamtfunktionalität dar, verdeutlichen jedoch die Funktionsweise des Systems.

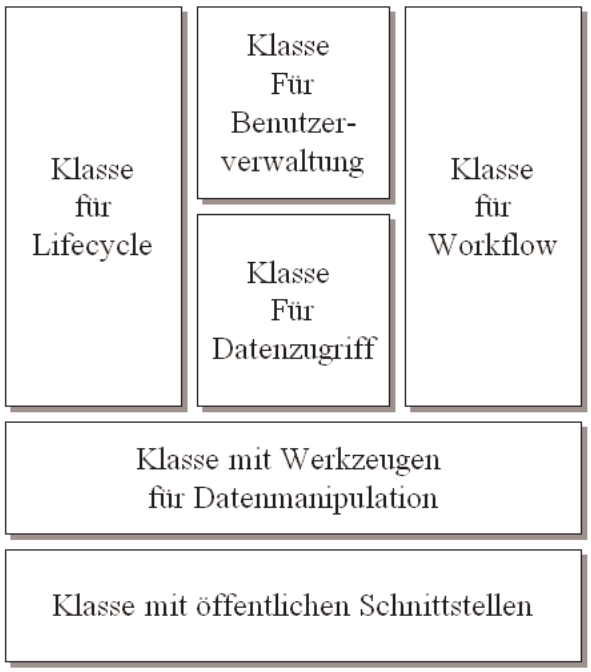


Abbildung 5.20: Klassen des SLM-Servers

Der SLM-Server setzt sich aus verschiedenen Komponenten zusammen, welche die unterschiedlichen Aufgaben des SLM-Systems abbilden. Den Kern des Systems bilden hierbei die Datenzugriffsfunktionen, welche die Kommunikation mit der Datenbank herstellen. Um sie herum sind die einzelnen Module angeordnet, die für die Verwaltung der

Benutzer, den Lebenszyklus, den Workflow und die öffentlichen Schnittstellen zuständig sind (vgl. Abbildung 5.20). Darüber hinaus ist eine Werkzeugklasse vorgesehen, welche die Geschäftslogik implementiert.

5.4 Client

Der Client des SLM-Systems wird durch den Anwender in der Regel explizit gestartet. Wenn allerdings über eines der integrierten Programme auf eine Funktionalität des SLM-Systems zugegriffen werden soll, erfolgt ein impliziter Start. Er wird auf jedem Client-Rechner jeweils nur einmal gestartet und bleibt bis zu seiner expliziten Beendigung durch den Benutzer oder dessen Abmeldung aktiv, unabhängig davon, ob ein spezifischer SLM-Server läuft. Der Client hat die Aufgabe, als Schnittstelle zwischen dem Benutzer, dem SLM-Server und den einzelnen integrierten Programmen zu dienen.

5.4.1 Benutzerführung

Der Client beinhaltet die Benutzerschnittstelle des SLM-Systems zum Benutzer. Er bietet dem Benutzer clientseitig die Sicht auf die Metadaten. Die Sicht auf die Daten selbst geschieht über die in das System integrierten Werkzeuge bzw. über das Betriebssystem und die dort installierten Programme. Soll also z.B. ein Word-Dokument oder eine PDF-Datei angezeigt werden, so muss auf dem Client ein geeigneter Viewer installiert sein. Das Anzeigen der Inhalte verschiedenster Dateiformate ist zunächst nicht die Aufgabe des SLM-Clients, da geeignete Viewer[24] von Drittanbietern zumeist kostenlos bereitgestellt werden oder diese hinzugekauft werden können.

Das Arbeiten mit dem SLM-System erfolgt projektorientiert (vgl. 5.2.1). Nach der Anmeldung an das SLM-System muss also zunächst ein Projektkontext hergestellt werden, d.h. der Benutzer wählt dasjenige Projekt aus, welches er bearbeiten bzw. ansehen möchte (vgl. Abbildung 5.21). Dieser Projektkontext bleibt während der gesamten Benutzersitzung bestehen, bis der Benutzer sich abmeldet oder entweder explizit einen anderen Projektkontext herstellt.

[24] Programme, die nur zur Ansicht von Dokumenten in bestimmten Formaten dienen

Abbildung 5.21: Projektdialog

Der Benutzer lädt jeweils das gesamte Projekt, mit dem er arbeiten möchte, zur Ansicht herunter. Dies ist notwendig, da der Entwickler alle zugehörigen Dateien benötigt, um sinnvoll an den Projektdateien entwickeln zu können. Soll z.B. an einer komplexen C++-Entwicklung gearbeitet werden, so benötigt der Entwickler die sämtlichen zugehörigen Dateien zum Kompilieren und Linken der Ausführungsdatei und anschließend auch zum Testen des Quellcodes.

Die meisten Dateien werden bei einem Kompilier- und Link-Vorgang nicht geändert. Die Änderungen beschränken sich bei der Arbeit an einem Programm in der Regel auf wenige Dateien, wobei häufig iterativ vorgegangen wird. Das heißt, dass der Programmierer Änderungen an einer oder zwei Dateien vornimmt, im Rahmen eines C++-Projekts z.B. an einer Quelldatei und an der zugehörigen Header-Datei. Nach den Änderungen werden diese getestet und je nach Testergebnis werden die Änderungen fortgeführt oder es wird an einer anderen Stelle des Quellcodes weitergearbeitet.

In diesem Zusammenhang ist es jedoch unsinnig, sämtliche zum Projekt gehörenden Dateien durch einen Check-Out aus dem SLM-System zu sperren, da ohnehin nur wenige Dateien geändert werden. Andererseits würde das Arbeiten an dem zu modifizierenden Code unnötig erschwert, wenn vor jeder Änderung die spezifische Datei ausgecheckt werden müsste und nach jeder Änderung sofort wieder eingecheckt werden muss. Aus diesem Grunde wird hier die folgende Vorgehensweise vorgeschlagen und realisiert:

- Der Entwickler nimmt an den Quellcodedateien die spezifischen Änderungen vor, bis die Tests ergeben, dass die Änderung zum gewünschten Ergebnis geführt haben.
- Er checkt die Quellcodedateien aus, an denen er Änderungen vorgenommen hat. Der Client überprüft, ob die jetzt ausgecheckten Versionen noch dieselben sind, die auf den Client-Rechner zur Ansicht herunter geladen wurden. Sind sie das nicht, so muss vom Entwickler die zugehörige Dokumentation eingesehen werden, in welcher sein Kollege die vorgenommenen Änderungen protokolliert hat. Gegebenenfalls muss mit Unterstützung des zweiten Bearbeiters der Datei dessen Änderungen in die neue Version eingepflegt werden.
- Der Entwickler checkt nun die lokal geänderte Datei in das SLM-System ein. Dem Eincheckenden werden automatisch diejenigen Dokumente vorgelegt, die im SLM-System für den spezifischen Dokumenttyp festgelegt wurden. Sie werden nach Bearbeitung auch in das SLM-System eingecheckt und verlinkt.

Eine der wichtigsten Aufgaben des SLM-Systems ist das automatische Mitführen der Dokumentation, sowie die Unterstützung der Entwickler bei diesem Prozess. Dadurch, dass dem Entwickler beim Eincheckvorgang automatisch bestimmte Dokumente vorgelegt werden, die er auszufüllen hat, wird ihm die unmittelbare Verantwortung dafür abgenommen und er kann sich besser auf seine eigentliche Aufgabe konzentrieren. Durch diese Vorgehensweise wird der häufigste Fehler bei vielen Softwareentwicklungsprojekten vermieden, das Setzen der Priorität auf die eigentliche Entwicklung nach dem Motto „ich muss damit fertig werden“, bei gleichzeitiger geringer Beachtung von Planung und Dokumentation der Entwicklungstätigkeit.

Eine weitere wichtige Aufgabe des SLM-Systems ist der Workflow innerhalb eines Entwicklungsteams (vgl. Kapitel 3.3). Er wird dadurch bestimmt, dass nach Abschluss eines Vorganges durch einen Mitarbeiter die entsprechenden Dokumente an die nächste Stelle zur weiteren Bearbeitung geleitet werden. Durch regelmäßige Abfragen des Clients, werden anstehende Arbeitsschritte im Rahmen des Workflows an den Client übermittelt und über einen Dialog dem Benutzer angezeigt. Dieser kann dann die entsprechenden Dokumente übernehmen und weiterbearbeiten.

5.4.2 Anmeldung/Abmeldung am SLM-System

Der Anmeldevorgang wird durch den Start des SLM-Clients ausgelöst, entweder durch einen expliziten Start oder durch einen impliziten durch Aufruf einer Client-Funktion über eine Schnittstelle. Die Anmeldung geschieht über einen Dialog, der in der Funktionalität dem von Windows verwendeten angelehnt ist. Er beinhaltet im wesentlichen drei Felder:

- Login-Namen,
- Kennwort
- SLM-Server, an dem die Anmeldung geschehen soll.

Die Auswahl des Server erfolgt durch Vorgabe aus einer Konfigurationsdatei, in einer späteren Version auch durch das automatische Scannen des Subnetzes, in dem sich der Rechner des Clients befindet. Es auch möglich, den Server durch direkte Angabe des Ortes (seiner IP-Adresse) zu spezifizieren. Das An- und Abmelden erfolgt während einer Sitzung mit einem Client-Rechner jeweils nur einmal. Während der gesamten Sitzung bleibt der Rechner angemeldet, die Abmeldung erfolgt automatisch bei der Beendigung des Clients.

Das Suchen nach Dateien geschieht durch Filterung anhand der Metadaten, wobei zusätzlich noch eine Volltextsuche der gefilterten Dateien nach einer Kombination von Stichworten inklusive so genannter Wildcards[25] hinzugezogen werden kann. Die zur Verfügung stehenden Metadaten hängen von der Klasse des zu suchenden Objektes ab, aber es stehen immer die Metadaten der allgemeinen Objekt-Klasse zur Verfügung, von der sich alle anderen Objekt-Klassen ableiten.

5.4.3 Projekte auf den Client laden

Der Entwickler benötigt zum Arbeiten immer eine Kombination verschiedener Dateien, von denen er aber in der Regel nur einige wenige auf einmal bearbeitet. Als Projekt, bzw.

[25] Wildcard: Ersatzsymbol, welches verschiedene Zeichen repräsentieren kann. Z.B. steht „*“ im allgemeinen für eine beliebige Zeichenfolge.

Teilprojekt kann also diejenige Kombination von Verzeichnissen und Dateien angesehen werden, die für die Arbeit eines Entwickler notwendig sind. Ein Gesamtprojekt verschiedener Entwickler kann sich dabei auch aus mehreren Teilprojekten zusammensetzen, wobei ein Teilprojekt die für die Arbeit eines Entwicklers notwendigen Dateien repräsentiert und das Gesamtprojekt die Teilprojekte aller Entwickler des Entwicklungsteams.

Bei der Anmeldung an den SLM-Server wird vom Entwickler ein Projektkontext hergestellt, d.h. er wählt aus der Gesamtheit aller Projekte, die im SLM-System zur Verfügung stehen, dasjenige aus, an welchem er arbeiten möchte. Er wird nun durch den Client aufgefordert, den Ladeprozess zu starten, was er bestätigen kann oder auch nicht. Die Projektdateien werden daraufhin durch den SLM-Server in der notwendigen Verzeichnisstruktur in der Datenaustauschfreigabe des SLM-Servers zusammengestellt und dann auf den Client entweder in das im SLM-System festgelegte Verzeichnis auf dem Client verschoben oder alternativ auf einen durch den Benutzer vorgegebenen Pfad (vgl. Abbildung 5.21).

5.4.4 Dateien Ein-/Auschecken

Das ein- und auschecken von Dateien ist einer der häufigsten und elementarsten Vorgänge im SLM-System. Aufgrund des Konzept für das kooperative Arbeiten mit dem SLM-System, bei dem generell ein Teilprojekt, welches bei einem Entwickler in Arbeit ist, zur Ansicht auf den Client-Rechner herunter geladen ist, muss nach dem Auschecken einer Datei überprüft werden, ob diese bereits auf dem Client vorhanden ist. Wenn diese vorhanden ist, gibt es drei Möglichkeiten:

1. Es ist dieselbe Version, die auch im SLM-System vorliegt,
2. auf dem Client liegt eine vom Entwickler geänderte Version vor oder
3. auf dem Server liegt eine neuere Version vor, als diejenige, die zur Ansicht geladen wurde.

Die Überprüfung, ob es sich um dieselbe oder eine andere Dateiversion handelt, ist insofern problematisch, als das bei verschiedenen Vorgängen im Rahmen des Entwicklungsprozesses, wie z.B. bei einem Übersetzungsprozess, die Dateien durch das Übersetzungssystem

„angefasst“ werden, ohne sie jedoch zu verändern und dadurch die Metainformationen dieser Dateien im Dateisystem gegenüber denjenigen verändert wurden, die im SLM-System vorlagen. Daher wird zur Überprüfung der Gleichheit der Dateien eine Checksumme aus diesen gebildet, die zum Vergleich herangezogen wird. In Abhängigkeit der vorab genannten drei Möglichkeiten reagiert das SLM-System jeweils anders. Bei Möglichkeit

- **eins** ist das Herunterladen der Datei auf den Client unnötig, da der Status der Datei durch das SLM-System lediglich in „ausgecheckt“ geändert wird.
- **zwei** wird der Entwickler gefragt, ob die vorhandene Datei überschrieben werden soll, oder ob die Datei lediglich den Status „ausgecheckt“ erhalten soll. Dies hat den Sinn, dass der Entwickler dann die von ihm geänderte Version der Datei sofort wieder als neue Version in das SLM-System einchecken kann.
- **drei** wurde die Version im SLM-System von einem weiteren Entwickler verändert. Es kann nun entweder die im SLM-System vorhandene Datei auf den Client herunter geladen werden, welche die dort vorhandene Version ersetzt, oder aber es muss ein Abgleich der Versionen stattfinden, wenn der Entwickler bereits Änderungen an seiner Version vorgenommen hat, die er nun in das System einchecken will.

Das Einchecken einer Datei funktioniert durch Identifikation der Datei anhand ihres relativen Pfades innerhalb der Projekt-Verzeichnisstruktur. Während der Client in Betrieb ist, merkt er sich die Objektidentifikationen aller Dateien, die zur Ansicht oder ausgecheckt auf dem Client-Rechner vorhanden sind und kann so schnell auf die entsprechenden Objekte im SLM-System zurückgreifen. Für den Fall zwei (s.o.) kann der Benutzer eine Datei, die er zur Ansicht auf seinen Rechner geladen und modifiziert hat, sofort einchecken. Das System führt dann implizit einen virtuellen Auscheckvorgang durch und checkt die Datei anhand der gespeicherten Objekt-Id sofort wieder ein.

Sollte der Client aus irgendeinem Grund beendet worden sein und nach einer Neuanmeldung eine Datei in das SLM-System eingecheckt werden, so muss zunächst das zugehörige Objekt in der Datenbank identifiziert werden. Dies kann dem Benutzer kaum zugemutet werden, denn er weiß ja im Zweifelsfall nicht, welche Version der Datei er seinerzeit ausgecheckt hat. Daher müssen die Informationen über die auf dem Client-Rechner vorhandenen Dateien, die

entweder ausgecheckt sind oder zur Ansicht vorliegen, persistent auf dem Client-Rechner abgelegt werden. Hierzu dient die Dateiüberwachung.

5.4.5 Dateiüberwachung

Die Informationen über alle auf dem Client vorhandenen Dateien und deren Stati werden lokal auf dem Client-Rechner in einer kleinen Datenbank abgelegt. Diese Datenbank wird jeweils beim Start des Clients erstellt und bei dessen Beendigung wieder gelöscht. Hierbei wird zuvor überprüft, ob noch veränderte Dateien vorhanden sind, die einen Abgleich mit dem SLM-System benötigen. Sollte das SLM-System aus irgendeinem Grunde unplanmäßig beendet worden sein, so bleibt diese Datenbank vorhanden und wird beim nächsten Start des SLM-Clients gelesen. Dadurch kann der Client ohne Informationsverluste genau an der Stelle weiterarbeiten, an welcher der Absturz erfolgt ist. In dieser Datenbank werden alle Dateien, ihre Pfadnamen und die zugehörigen Objekt-IDs gespeichert, ebenso wie die Checksummen zur Überprüfung der Dateiversionen.

5.4.6 Integration

Die Integration der verschiedenen Werkzeuge mit dem SLM-System zu einer kundenspezifischen Entwicklungsumgebung geschieht wie bereits erklärt über den Client. Hierzu ist zwischen diesen Werkzeugen und dem Client ein bestimmter Datenaustausch notwendig, um zumindest einen Minimalumfang an Funktionalität zu implementieren. Um überhaupt mit dem SLM-System integriert werden zu können, müssen die zu integrierenden Programme die Möglichkeit besitzen, diese Minimalanforderungen zu erfüllen. Die Implementierung dieser Funktionalitäten geschieht über Schnittstellen, die entweder in den Client des SLM-Systems zu integrieren sind oder auch teilweise in das Werkzeug. Leider bieten nicht alle Werkzeuge die gleichen Möglichkeiten zur Schnittstellenprogrammierung und bei einigen sind diese nur sehr rudimentär vorhanden. Die Mindestanforderungen für eine Integration sind:

- Aufruf der Benutzerschnittstelle des Client vom integrierten Programm aus,
- Laden einer Datei ferngesteuert durch den Client im integrierten Programm,
- Speichern einer Datei ferngesteuert durch den Client aus dem integrierten Programm.

Folgende Arten von Schnittstellen stehen generell zur Verfügung:

- **API** (Application Programming Interface): Programmierschnittstelle einer Anwendung, auf die über das Einbinden einer DLL zugegriffen werden kann. Diese DLL kann dann auf bestimmte interne Funktionalitäten der Anwendung direkt zugreifen.
- **DDE** (Dynamic Data Exchange): Erster Ansatz für eine Interapplikationsschnittstelle für den Hauptspeicher basierten Datenaustausch. Bei der Herstellung einer Verbindung wird mit der Anwendung vereinbart, über welches „Thema" oder „Topic" gesprochen wird. Je nach „Thema" stehen dann verschiedene Funktionen zur Verfügung.
- **COM/COM+** (Component Object Model): Über die so genannte IDL (Interface Definition Language) werden spezifische Schnittstellen deklariert, die von jeder Anwendung aus aufgerufen werden können, wenn sie über das Betriebssystem registriert (Registry) sind. Praktisch das gesamte MS-Windows Betriebssystem, von Windows 95 bis hin zu Windows XP basiert auf dieser Technologie.
- .NET**-Assemblies**: Durch das Platzieren einer Assembly[26] im GAC[27] des .NET-Frameworks, können die im Manifest aufgelisteten Namespaces des Assemblies von jeder Anwendung importiert und somit alle dort zur Verfügung stehenden Funktionen aufgerufen werden [21] [16].

Welche dieser grundsätzlichen Arten für die Entwicklung einer Schnittstelle zu einem speziellen Werkzeug letztendlich verwendet wird, hängt in erster Linie davon ab, welche Programmierschnittstelle dieses Werkzeug zur Verfügung stellt. Grundsätzlich kann das SLM Schnittstellen auf Basis aller beschriebenen Architekturen realisieren. Im Folgenden werden

[26] Sammlung von DLL- und EXE-Dateien, die den Code enthalten mit einem Manifest mit den Metadaten des Assemblies, in welchem alle enthaltenen Funktionalitäten deklariert werden.

nun die Schnittstellen zu den Werkzeugen beschrieben, die im Rahmen der Basisfunktionalität mit dem SLM integriert werden.

5.4.7 Schnittstellen zu CASE-Tools

Mit CASE-Tools werden im Prinzip alle Werkzeuge bezeichnet, die den Softwareentwicklungsprozess in irgendeiner Weise unterstützen, also auch letztendlich alle Werkzeuge die irgendwie für den Computer verständliche Programme und Daten erzeugen (also auch Compiler, Linker), sowie dem Benutzer Hilfestellung dabei leisten (also z.B. auch Editoren mit Syntax-Highlighting[28]), denn im Prinzip könnte der Anwender ja auch direkt Maschinencode schreiben und in den Rechner eingeben. Im engeren Sinne werden hier aber häufig Systeme verstanden, die es dem Anwender erlauben, aus der Modellierung der Daten, Klassen und Funktionalitäten mit Hilfe abstrakter und teilweise auch graphischer Beschreibungsmöglichkeiten direkt ein Anwendungsgerüst in einer spezifischen Programmiersprache automatisch zu erstellen. Zu diesen CASE-Tools zählen z.B. Rational Rose [27] oder MS-Visio, aber auch eine Entwicklungsumgebung wie MS-Visual Studio oder IBM Visual Age, die ja über Assistenten auch direkt komplexe Anwendungsgerüste erstellen können (vgl. Kapitel 4.4).

Diese Systeme können die erzeugten Daten in verschiedener Weise speichern und laden. In der Regel in einem spezifischen eigenen Datenformat, in der Universal Meta Language (UML) und in den zu erzeugenden Daten und Programmstrukturen. Für die Integration in den Client genügen hier zunächst die oben beschriebenen Funktionalitäten, also Aufruf der Client-Benutzerschnittstelle, ferngesteuertes Laden und Speichern von Dateien.

[27] Global Assembly Cache

[28] Verschiedene Sprachelemente werden durch Einfärbung für den Benutzer schneller sichtbar (z.B. Schlüsselworte, Texte, Kommentare usw.)

5.4.8 Schnittstellen zu Office-Software

Office-Software hat im Zusammenhang mit dem SLM-System im wesentlichen die Aufgabe, Metadokumente anzuzeigen und zu erzeugen. Hierzu gehören z.B. Textdokumente, wie MS-Word oder auch Listen- und Tabellendokumente wie MS-Excel. Aber auch andere Produkte, wie z.B. für die Projektplanung (MS-Project) gehören dazu. Es genügen daher auch hier Funktionalitäten zum Anzeigen, Ein- und Auschecken in das SLM-System.

5.4.9 Schnittstellen zu Code-Ausführungssoftware

Die Code-Ausführungssoftware ist ein elementarer Bestandteil jeder Entwicklungsumgebung, denn ohne sie ist die Softwareentwicklung nicht möglich. Im Prinzip existieren zwei verschiedene Arten von Code-Ausführungssoftware: Ein Compiler, welcher die Software von einer so genannten Hochsprache in Maschinencode übersetzen und direkt auf der entsprechenden Hardware ausgeführt werden kann und der Interpreter, der die Software in ihrer Hochsprachenform ausführt, also praktisch ein virtueller Prozessor für diese Hochsprache. Beispiele für

- Compilersprachen sind: C/C++, Pascal, ADA, Fortran
- Interpretersprachen sind: BASIC, JAVA, LISP, CAD-Makrosprachen

Häufig werden heute jedoch beide Technologien kombiniert, um den Hauptnachteil von interpretiertem Code, nämlich die vergleichsweise langsame Ausführungsgeschwindigkeit zu umgehen, die etwa um Faktor 10 unter der von kompiliertem Code liegt. Dies geschieht, indem beim Ausführungsstart der zu interpretierende Code durch einen Just-In-Time-Compiler im Maschinencode übersetzt wird.

Zusätzlich zum Einchecken, Auschecken und Anzeigen sind nun noch Schnittstellen zum Ausführen und Testen der Software notwendig. Im Falle der Compilersprachen ist als Minimalvoraussetzung der Start des Compilers, ggf. des Linkers, der die eigentliche Ausführungsdatei aus verschiedenen kompilierten Komponenten zusammensetzt und

letztendlich der Aufruf eines so genannten Debuggers[29] mit der erzeugten Ausführungsdatei, der das zeilenweise Durchgehen der Ausführungsdatei zur Fehlersuche ermöglicht. Für die Interpretersprachen sind zwei Fälle zu unterscheiden:

1. Es handelt sich um einen allein stehenden Interpreter, wie z.B. Visual Basic, der in das Betriebssystem des Host-Rechners integriert wird. In diesem Fall ist nur ein einfacher Ausführungsbefehl für die entsprechende Quellcodedatei an das Betriebsystem abzusetzen. Für den Debugger gilt das Gleiche wie für die Compilersprache.

2. Der Interpreter ist in ein Drittprogramm integriert, wie z.B. eine Officeanwendung oder ein CAD-System. In diesem Fall muss eine Schnittstelle zu dem jeweiligen Drittprogramm hergestellt werden, der das ferngesteuerte Laden und Ausführen ermöglicht, ebenso wie das Ansteuern von Debugmöglichkeiten, die je nach dem auch nur sehr rudimentär sein können oder gar nicht vorhanden sind.

In dem in Kapitel 6 vorgestellten Realisierungsbeispiel handelt es sich um Fall Nr. 2. Hier ist die Code-Ausführungssoftware das CAD-System CoCreate ME10.

[29] Entwanzer; Programm zum Finden von Fehlern in einer Software

6 Beispielhafte Anwendung des SLM-Konzeptes

Für den Einsatz des SLM-Systems in der Praxis existieren vielfältige Anwendungsbereiche. Allein aus Zeitgründen lässt sich im Rahmen dieser Arbeit die gesamte Breite der Anwendungsmöglichkeiten des SLM-Systems nicht realisieren. Für die Durchführung von derart komplexen Softwareentwicklungen ist zumindest ein Entwicklungsteam erforderlich. Das heißt, auch wenn hier ein sehr gutes Programm zur Unterstützung des Softwareentwicklungsprozesses vorliegt, kann eine einzelne Person große Softwareentwicklungen nicht in einem vertretbaren Zeitraum bewältigen. Aus diesem Grund soll für den Nachweis der Machbarkeit des hier konzipierten SLM-Systems ein Software-Produkt ausgewählt werden, welches die gesamte Breite des Konzeptes erfordert. Im Prinzip lässt sich das SLM-System für jedes beliebige Softwareentwicklungsprojekt konfigurieren, jedoch existiert in manchen Bereichen der Softwareentwicklung zumindest eine Unterstützung für den Zeitraum der eigentlichen Softwareentwicklung, weniger für den Zeitraum der danach folgt, wie Weiterentwicklung und Support. Diese Bereiche sind für den Nachweis weniger interessant.

Eines der wichtigsten Einsatzgebiete des SLM-Systems ist deshalb in den Bereichen gegeben, in denen es bisher noch keine Unterstützung für den gesamten Softwareentwicklungsprozess gibt. Hierzu gehören unter anderem anwendungsspezifische Makroprogramme, die im Prinzip nur für kleinere firmenspezifische Anpassungen vorgesehen sind. Zu diesen firmenspezifischen Lösungen gehören zweifelsfrei die Anwendungen der Makroprogrammierung aus dem CAD-Bereich, wo häufig genutzte Arbeitsabläufe, wie z.B. die Erstellung bestimmter Normteile als Makrocode abgelegt wird, um den Konstrukteuren die Arbeit zu erleichtern. Auch bestimmte firmenspezifische Berechnungs- und Auslegungsalgorithmen werden in derartigen Makrocodeumgebungen implementiert.

6.1 Auswahl eines Anwendungsbeispiels

Bei manchen Softwarelieferanten sind im Laufe der Zeit auf Grund von Kundenwünschen aus einer Vielzahl solcher kleinen Anpassungen recht komplexe Software-Produkte entstanden.

Diese sind aufgrund der fehlenden Unterstützung des hier vorliegenden Entwicklungsprozesses nur sehr aufwändig zu warten und zu erweitern. Für die jeweiligen vertreibenden und auch anwendenden Unternehmen sind diese so genannten Software-Produkte von großer Wichtigkeit, da gerade in diesen speziellen Entwicklungen häufig das gesamte Know-How vieler Personen und Jahre steckt. Wenn allerdings für diese entsprechende Software weder eine Entwicklungsumgebung noch eine Testumgebung existiert, das Software-Produkt nicht dokumentiert ist und alle ursprünglichen Entwickler nicht mehr verfügbar sind, wird dies sowohl für den Lieferanten als auch für den Anwender dieses Produktes höchst problematisch.

Dieser Fall tritt gerade in jüngster Zeit immer häufiger bei Softwareentwicklungen auf, die recht früh begonnen wurden und bei denen seinerzeit nicht mit der notwendigen Ernsthaftigkeit auf eine ausreichende Dokumentation geachtet wurde. Letzteres geschah eigentlich erst mit Beginn der objektorientierten Softwareentwicklung. Mit zunehmender Verbreitung dieser Software-Produkte wurden die Wünsche der Anwender immer umfangreicher, was letztlich von den Software-Lieferanten bzw. –Entwicklern nicht mehr gewährleistet werden kann. In der Regel hilft hier nur noch ein Redesign des gesamten Software-Produktes.

Einen derartig typischen Anwendungsfall stellt das Makro-Programm ProfiPlus dar, welches unter dem Produktnamen **PP2000** von einem mittelständischen CAD-Dienstleister vertrieben wird. Hierbei handelt es sich um eine Erweiterung des Funktionsumfanges des 2D-CAD-Systems ME10 von CoCreate, d.h., PP2000 ersetzt die Standard-Benutzeroberfläche von ME10 und beinhaltet darüber hinaus eine Vielzahl von Funktionen, die nicht in ME10 enthalten sind. Die Entwicklung von PP2000 begann in den neunziger Jahren. Die ursprünglichen Entwickler von PP2000 sind heute nicht mehr verfügbar. In der Zwischenzeit wurden von verschiedenen Stellen Änderungen und Erweiterungen an PP2000 vorgenommen.

Da in den Quelltexten von PP2000 nur sehr selten Kommentare zu finden sind, ist eine Einarbeitung in das System für jeden Außenstehenden äußerst schwierig und langwierig. Erschwerend kommt hinzu, dass das System heute aufgrund seines Lebenszyklus eine erhöhte Fehleranfälligkeit aufweist. Das Programm wurde zum überwiegenden Teil in ME10-Makrosprache erstellt, wobei die Grundsätze der strukturierten Programmierung oft nur

unzureichend berücksichtigt wurden. Alle diese Punkte führen in der Summe dazu, dass eine Fehlerbereinigung oder eine Erweiterung des Funktionsumfanges nur mit einem nicht überschaubarem Aufwand durchgeführt werden kann. Nutzen und Aufwand stehen in keinem akzeptablen Verhältnis mehr [28].

Diese Tatsache führt zu dem Schluss, dass nur ein komplettes Redesign des Produktes PP2000 unter Beibehaltung der vorhandenen Funktionalitäten und der hier vorgegebenen Benutzeroberfläche auf der Basis einer neuen strukturierten Softwarearchitektur zu einem stabilen und erweiterungsfähigen Produkt führen kann, welches auch in Zukunft den Ansprüchen des breiten Kundenkreises von PP2000 genügen wird.

Diese Forderungen bildeten auch die Grundlage für die Konzipierung des hier vorgestellten SLM-Systems als Entwicklungsumgebung für die Durchführung von Softwareentwicklungsprozessen. Obwohl das SLM-System produktunabhängig konzipiert wurde, so verlangt eine gesicherte Machbarkeit des Konzeptes die ständige Überprüfung desselben durch eine jederzeitige Rückkopplung mit vorhandenen Softwareentwicklungen. Bevor nun das neue ProfiPlus (**PPNEU**) hier vorgestellt wird, soll zunächst der IST-Zustand des momentan verfügbaren ProfiPlus (PP2000) analysiert und daraus die Forderungen an das System PPNEU formuliert werden.

6.2 IST-Zustand der Softwareentwicklung PP2000

Das Softwarepaket PP2000 ist überwiegend ein in Makrocode geschriebenes Produkt, welches die Benutzeroberfläche des 2D-CAD-Systems ME10 anwenderfreundlicher gestaltet und darüber hinaus viele Werkzeuge zur Unterstützung des Zeichnungsprozesses beinhaltet, wie die Erstellung bestimmter Normteile, die automatische Generierung von abgesetzten Wellen, Bohrbildern und Stücklisten oder die Unterstützung von vielen anderen Routineaufgaben, denen sich der Konstrukteur tagtäglich stellen muss, wie etwa das batchgesteuerte Plotten von Einzelteil- und Baugruppenzeichnungen.

Die standardmäßige textorientierte Benutzerführung von ME10 weist (vgl. Abbildung 6.1) neben der Status- und Eingabezeile (a) und der Zeichnungsfläche (b) für die Kommunikation

eine Vielzahl von Einzelmenüs für die weitere Bearbeitung auf. Diese Menüs sind als Wechselmenüs gestaltet und ermöglichen so u.a. das Erstellen (c), Bemaßen, Ändern und Speichern (d) von Zeichnungen. Die Handhabung wird durch weitere Menüs erleichtert, welche die häufig benutzten Befehle (e) und Zusatzmodule (f) wie die Nutzung von Normteilen enthalten. Zur Unterstützung der Konstruktionstätigkeiten sind darüber hinaus eine Vielzahl von Hilfsfunktionen (g,h) angeordnet wie z.B. Interrupt- (Darstellung und Auswahl), Fenster- und Toolbox-Funktionen.

Durch den täglichen Umgang mit ME10 hat sich bei der Anwendung herausgestellt, dass die hier vorliegende Benutzerführung nicht optimal gestaltet ist und des weiteren einige wichtige Funktionalitäten fehlen, welche die Arbeit mit ME10 wesentlich erleichtern würden. Diese Wünsche seitens der Anwender führten zu der Entwicklung von PP2000. Mit der Entwicklung von PP2000 wurde nach und nach die vorhandene textorientierte Benutzeroberfläche zu ca. 95% durch eine grafische Benutzeroberfläche ersetzt und um die zusätzlich notwendigen Funktionalitäten erweitert.

Sehr gut lässt sich an dem Befehls-Menü „ERSTELLEN“ die bessere Handhabbarkeit erkennen (vgl. Abbildung 6.1). Insbesondere die grafischen ICONs zeigen dem Anwender auf einen Blick, welche der Geometriefunktionen sich hinter den jeweils ausgewählten Befehlen verbergen. Des weiteren wird die Arbeitsweise für den Benutzer durch das Zusammenfassen von Befehlen aus verschiedenen Menüs in ein Menü wesentlich erleichtert. Dies gilt z.B. für den Befehl „LINIENART“ (in Abbildung 6.1), der hier in das Menü „ERSTELLEN“ mit eingebunden wurde, während dieser im Standardmenü von ME10 in dem Hilfsmenü (g) angeordnet ist.

Aus der Abbildung dieser neu gestalteten Benutzeroberfläche lässt sich allerdings nicht die Komplexität des Softwarepaketes PP2000 erkennen. Auf Grund der langen Entwicklungszeit (ca. 10 Jahre), welche durch eine Vielzahl ungesteuerter Einzelentwicklungen geprägt ist, setzt sich heute das Gesamtpaket PP2000 aus vielen Einzelkomponenten zusammen, die bezüglich der Wirksamkeit der einzelnen Funktionalitäten mit unterschiedlicher Tiefe gegenseitig voneinander abhängig sind. Gerade diese Abhängigkeiten bei gleichzeitig fehlender Dokumentation erschweren eine Erweiterung und Wartung dieses Software-Produktes erheblich. Darüber hinaus wurde das Projekt überwiegend unter der Regie eines einzigen Mitarbeiters durchgeführt, der somit die vollständige Übersicht über das Projekt hatte und der heute nicht mehr verfügbar ist.

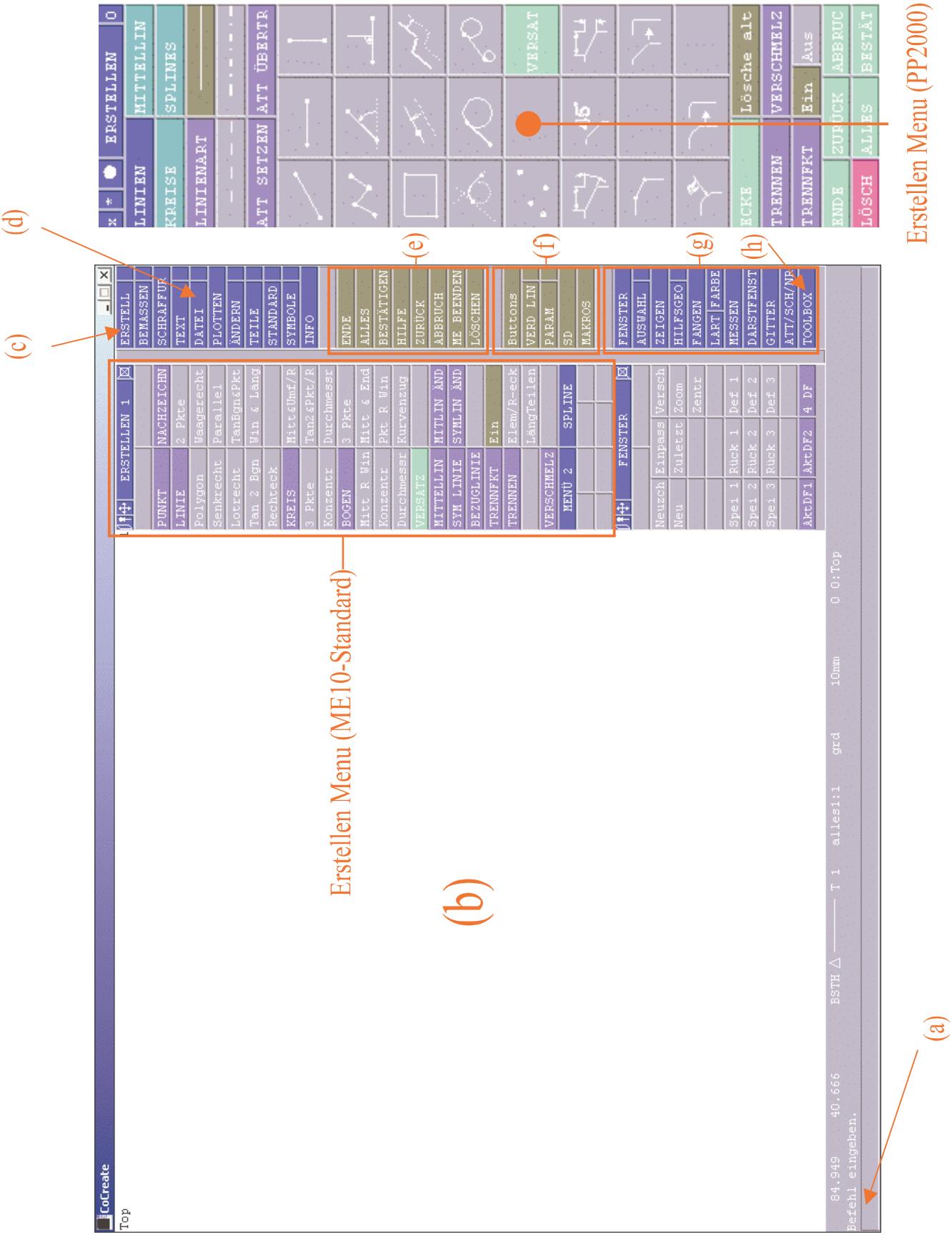


Abbildung 6.1: Standardoberfläche von ME10 und ProfiPlus-Erweiterung.

Insgesamt setzt sich das PP2000 aus etwa 800 Dateien zusammen und enthält ca. 4500 ME10-Makros und zuzüglich eine Vielzahl von C-Quelltexten und Bibliotheken (vgl. Abbildung 6.2).

Die dargestellten Rechtecke sind als Verzeichnisse zu verstehen, die jeweils eine Vielzahl einzelner Dateien enthalten, hier dargestellt durch Pfeile mit Dateiname (-> Bohrpgew.dat). Des weiteren muss hier noch festgestellt werden, dass auf Grund des fehlenden Managements einzelne Makrodateien die gleichen Makros mit unterschiedlichen Inhalten enthalten. Das heißt, zu irgendeinem Zeitpunkt wurde ein Makro mit einer verbesserten Funktionalität neu erstellt und aus falsch verstandener Sicherheit an irgendeiner anderen Stelle in eine der vorhandenen Makrodateien eingebunden. Dies bedeutet für die Nutzung, dass immer nur das jeweils zuletzt eingelesene Makro bei einem Neustart des Softwarepaketes genutzt wird, was aber nicht immer das aktuelle Makro sein muss.

Des weiteren muss hier noch auf eine Besonderheit hingewiesen werden. Da die Makrosprache eine interpretierende Sprache ist und somit die Quelltexte lesbar im ASCII-Format vorliegen, müssen die benutzerspezifischen Makros vor ihrer Auslieferung an den Benutzer (Kunden) aus Sicherheitsgründen (der Benutzer darf diese nicht verändern) verschlüsselt werden. Lediglich die Makros mit den Menu-Definitionen werden unverschlüsselt ausgeliefert, damit der Kunde sie ggf. anpassen kann. Dies bedeutet, fast alle Makros müssen einmal als Quelltext für den Entwickler in unverschlüsselter und als Kundenmakros in verschlüsselter Form vorliegen (vgl. Abbildung 6.2). Dies verlangt einen zusätzlichen Managementaufwand.

Die Verzeichnisstruktur von PP2000 besitzt eine hohe Komplexität (vgl. Abbildung 6.2). Der obere Teil zeigt die Struktur der Verzeichnisse, Unterverzeichnisse und der darin enthaltenen Daten in der Form, wie sie auch auf einem Kundenrechner installiert sind. In dem Unterverzeichnis „**Makros**“ unterhalb des Hauptordners befinden sich alle verschlüsselten Makodateien, aus denen die PP2000-Software besteht. In den Verzeichnissen „**dll**“ respektive „**sl**“ befinden sich die Bibliotheksdateien, die den Umfang von ME10 unter Windows (**dll**), beziehungsweise Unix (**sl**) um die Funktionalitäten erweitern, die sich nur schlecht oder gar nicht in einen ME10-Makrocode umsetzen lassen.

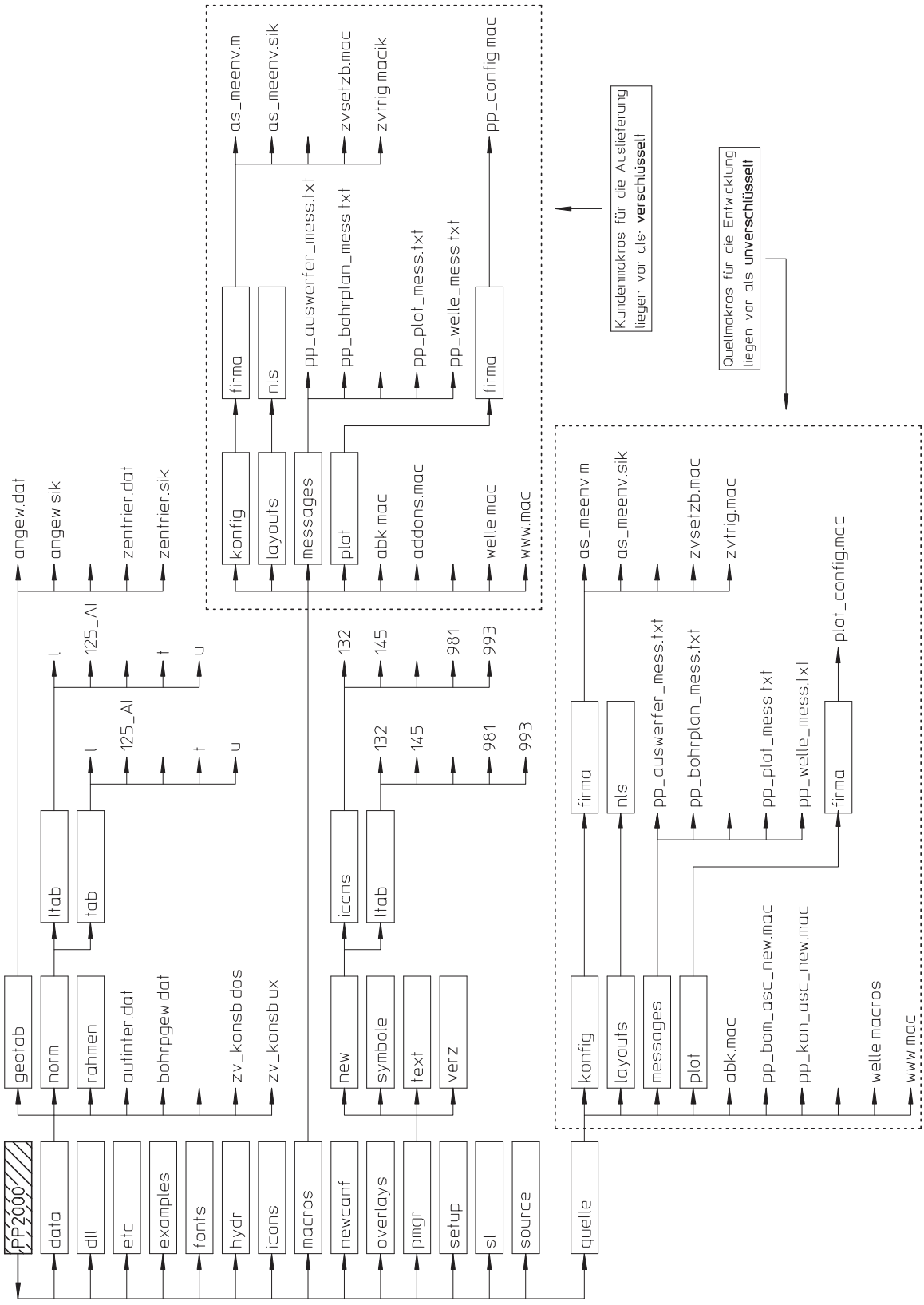


Abbildung 6.2: Verzeichnisstruktur von PP2000 (ausschnittsweise)

Alle anderen Verzeichnisse enthalten im wesentlichen nur Konfigurationsdateien, Tabellen für Normteile oder Symbole für die Anzeige in ME10, jedoch keinen weiteren Programmcode. In der Entwicklungsversion enthält das Hauptverzeichnis zusätzlich noch den Ordner „**Quellen**“, der in Aufbau und Struktur dem Ordner „**Makros**“ entspricht, jedoch alle Makrodateien in unverschlüsseltem, lesbarem Textformat enthält, so dass sie von den Entwicklern bearbeitet werden können. In diesem Verzeichnis sind auch die oben genannten Dateien enthalten, die verschiedene Makros redundant enthalten. Die Quellen für die Bibliotheken werden in separaten Ordnern verwaltet und sind hier nicht abgebildet.

Für eine ordnungsgemäße Initialisierung des Makropaketes muss der Startprozess der ME10-Software angepasst werden, was über Betriebssystem-Skripte [29] geschieht. Wie bereits erläutert, sind einige dieser Makros zum Teil mehrfach in verschieden Versionen in den Makrodateien vorhanden, was dazu führt, dass die Reihenfolge in der die Makros eingeladen werden, über die momentan aktuelle Version entscheidet. Aufgrund dieser Tatsache lässt sich die genaue Anzahl der verschiedenen Makros nur nach einer Bereinigung der Liste angeben, die aber aufwändig ist. Die Startmakros sind teilweise mehrfach verschachtelt, d.h. dass im Startmakro andere Makros geladen werden, in denen wiederum weitere Makros geladen werden. Zudem sind in den Startmakros das Laden, die Ausführung und die Konfiguration des Systems teilweise miteinander vermischt. Ein weiteres Problem, insbesondere für die Serviceingenieure vor Ort, ist durch die unangenehme Eigenschaft von ME10 gegeben, dass beim Fehlschlag des Ladens eines Makros oder Moduls der gesamte Startprozess zunächst fortgesetzt wird und erst bei dessen Abschluss der Fehler zutage tritt. Hierbei ist nicht ersichtlich, welche Datei nicht geladen werden konnte oder nicht vorhanden war, so dass der Serviceingenieur nun aufwändig nach dem Problem suchen muss.

Einige Makrodateien werden erst auf Interaktion des Benutzers nachgeladen, so dass die entsprechenden Makrodateien zu Systemstart nicht im Speicher vorhanden sind. Dies beschleunigt zwar im Prinzip den Programmstart und minimiert den Speicherplatzbedarf, erhöht aber sehr stark die Unübersichtlichkeit für die Entwickler.

Zum besseren Verständnis für die Anwendung des SLM-Systems sei hier noch auf die momentane Vorgehensweise für die Nutzung des Systems PP2000 insbesondere durch die Entwickler eingegangen. Für die Weiterentwicklung an dem System PP2000 wird immer eine vollständig funktionierende Umgebung benötigt, da die meisten Funktionalitäten des

Softwarepakets mit anderen Programmteilen interagieren und teilweise einzelne Module Funktionen aus anderen Modulen benötigen.

Für den Startvorgang von PP2000 muss in der Custom.mac-Datei, die sich im Programmverzeichnis von ME10 befindet, die Startdatei von PP2000 eingetragen werden. Diese wird automatisch beim Startvorgang von ME10 geladen und damit der gesamte Start- und Initialisierungsvorgang von PP2000 durchgeführt (vgl. Abbildung 6.3). Wird PP2000 in einer so genannten Testumgebung gestartet, benötigt der Entwickler an Stelle der verschlüsselten Kundenmakros die unverschlüsselten Makros. Dies ist zum einen erforderlich, da nicht nach jedem Entwicklungsschritt erneut verschlüsselt wird, zum anderen werden für Testzwecke die unverschlüsselten Makros benötigt, da nur die Abarbeitung dieser unverschlüsselten Makros von ME10 protokolliert werden kann. Das Erstellen einer Protokolldatei ist das einzige Hilfsmittel für die Fehlersuche, welches in ME10 zur Verfügung steht. Dabei kann mit Hilfe der „Trace“-Funktion ein Ausführungsprotokoll in eine separate Datei geschrieben werden, um die Ausführungsreihenfolge des Codes und den Inhalt der Variablen zum Ausführungszeitraum zu ermitteln.

Prinzipiell könnte in der Startreihenfolge für PP2000 von den Entwicklern auf das Laden der verschlüsselten Kundenmakros verzichtet werden, aber leider liegen nicht alle Makros in unverschlüsselter Form vor, d.h., der PP2000-Quellcode liegt nicht mehr vollständig vor.

Wenn PP2000 im Speicher des CAD-Programms ME10 vollständig in unverschlüsselter Form vorliegt, kann nun ein iterativer Entwicklungsvorgang gestartet werden. Hierzu werden vom Entwickler zur Bearbeitung einer Funktionalität in PP2000, die in der Regel durch das Zusammenspiel mehrerer Makros abgebildet wird, die entsprechenden Quelldateien mit einem Texteditor bearbeitet (vgl. Abbildung 6.14). Nach der Modifikation müssen die entsprechenden Quellcodedateien gespeichert werden und manuell in das ME10 nachgeladen werden. Um nun im Zusammenspiel aller Makros die entsprechende Funktionalität zu testen, kann dies durch einfaches Aktivieren der Funktionalität oder im Falle des Nichtfunktionierens durch die Analyse eines entsprechenden Ausführungs-Protokolls geschehen. Wenn die Funktionalität in PP2000 zufrieden stellend implementiert wurde, kann der Entwickler nun entweder an weiteren Funktionalitäten arbeiten, oder nach Abschluss seiner Arbeit die für die Installation beim Kunden notwendigen verschlüsselten Versionen erstellen.

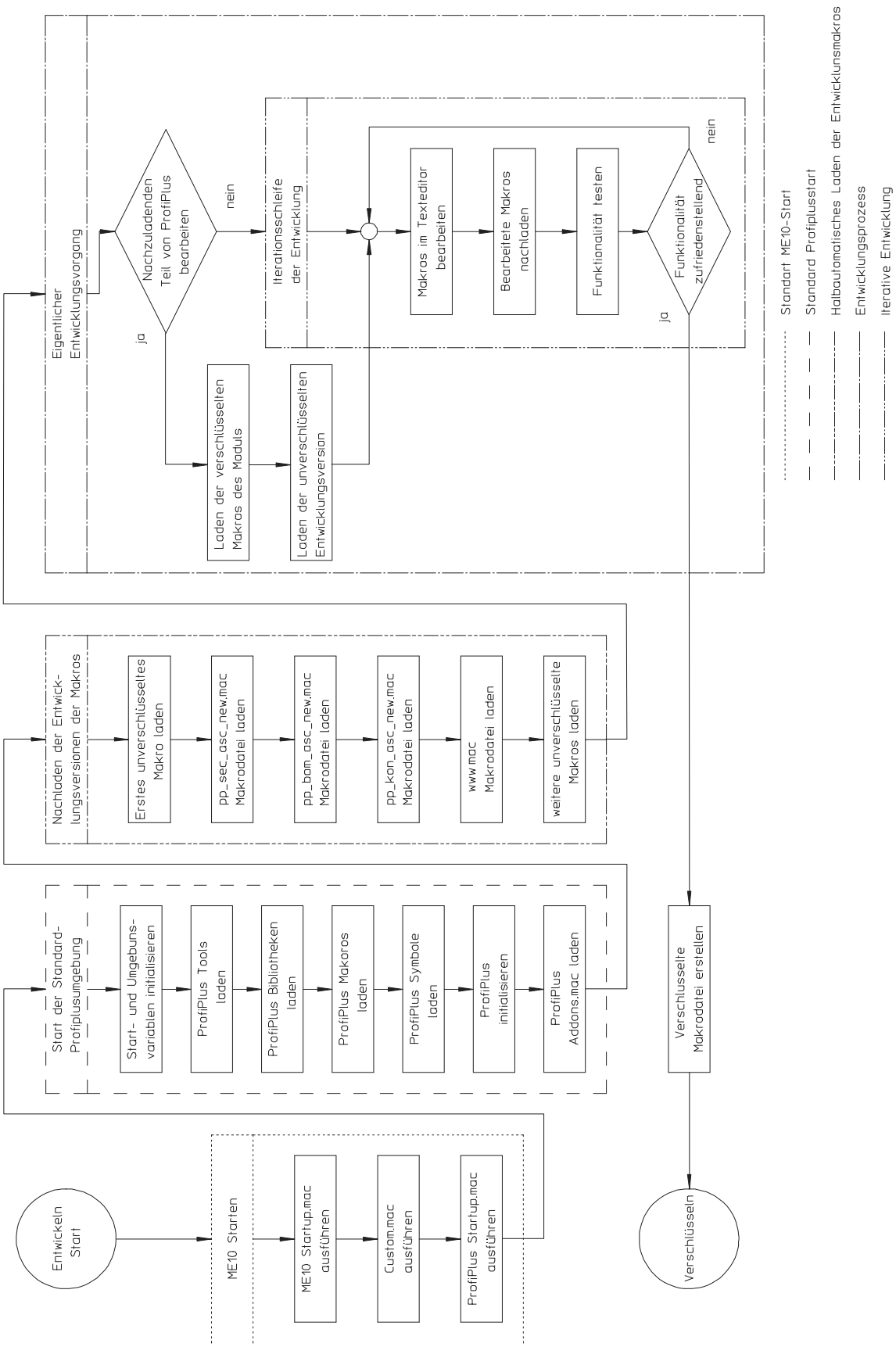


Abbildung 6.3: Entwicklung an PP2000 (bisherige Vorgehensweise).

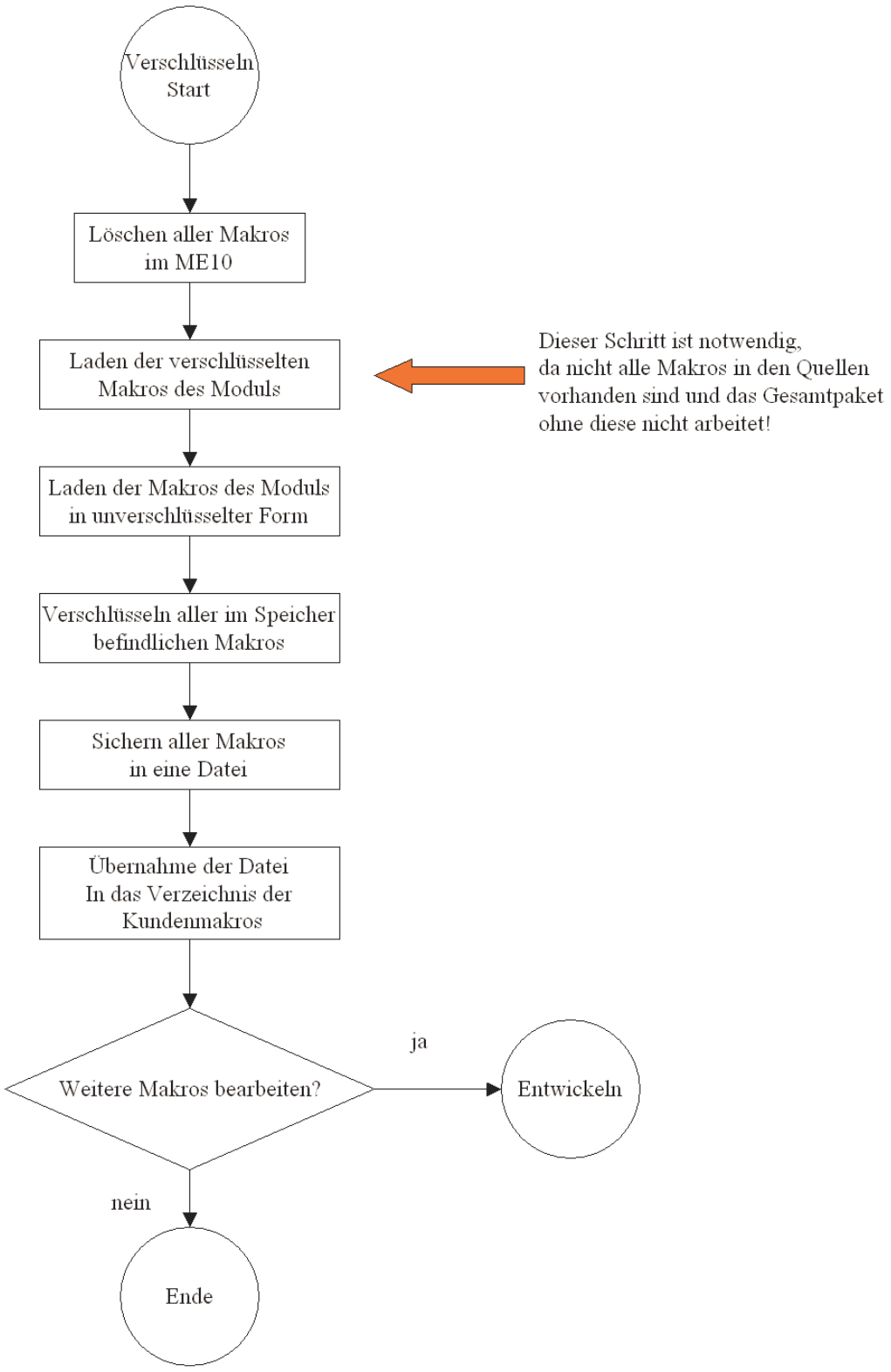


Abbildung 6.4: Vorgehensweise bei der Verschlüsselung von Kundenmakros

Das Verschlüsseln der Makros in ME10 erfolgt entweder für die einzelnen Makros sukzessiv nacheinander oder für alle Makros gleichzeitig. Diese verschlüsselten Makros lassen sich nun entweder einzeln in verschiedene Dateien speichern oder alle im Speicher befindlichen Makros in eine Datei. Bei der Verschlüsselung der Makrodatei muss sichergestellt werden, dass sich im Speicher nur die Makros befinden, die später in der Makrodatei enthalten sein sollen (vgl. Abbildung 6.4).

Zunächst müssen alle Makros aus dem Speicher von ME10 entfernt werden. Nach dem Löschen aller Makros ist ME10 nur noch über die Kommandozeile zu bedienen, da die Oberfläche von ME10 ebenfalls in Makrocode geschrieben ist. Danach werden die alten, verschlüsselten Makros des Moduls manuell nachgeladen, um sicherzustellen, dass eventuell fehlende Makros aus den Quellcodes auch in den neuen verschlüsselten Dateien vorhanden sind. Als nächstes werden nun die unverschlüsselten, bearbeiteten Makros manuell hinzu geladen, welche die verschlüsselten Makros ersetzen sollen. Diese können nun über einen einfachen Befehl in ihrer Gesamtheit verschlüsselt und gemeinsam in eine verschlüsselte Makrodatei geschrieben werden. Diese verschlüsselte Makrodatei kann nun in dem Unterverzeichnis mit den Kundenmakros die alte dort vorliegende Version ersetzen. Wenn nun die Weiterentwicklung fortgesetzt werden soll, so muss ME10 beendet und der gesamte Vorgang mit dem Neustart von ME10 wiederholt werden.

Die Beschreibung verdeutlicht, dass der Entwicklungsprozess von PP2000 aufgrund gewisser Unzulänglichkeiten von ME10 relativ umständlich ist. Die wesentlichen Unzulänglichkeiten hierbei sind, dass es für den ME10-Makrocode keine geeignete Entwicklungsumgebung gibt, die den Programmierer in seiner Arbeit unterstützt. Makros können zwar auch prinzipiell direkt im ME10 editiert werden, dies ist aber noch umständlicher und unübersichtlicher als mit einem Texteditor. Weiterhin sind die Debug-Funktionen im ME10 als mangelhaft zu bezeichnen, da die Ausführungs-Protokolle nur schlecht zu lesen sind und von den Entwicklern bei deren Interpretation eine erhebliche Transferleistung verlangt wird. Diese Mängel rühren in erster Linie daher, dass die ME10-Makrosprache für kleine Automatisierungen und Anpassungen im ME10 vorgesehen ist und nicht für die Erstellung einer so komplexen Software wie PP2000.

So lange immer nur ein Entwickler an PP2000 arbeitet, ergeben sich keine Probleme hinsichtlich der Konsistenz des Codes. Erst wenn mehr als ein Entwickler gleichzeitig tätig wird, ergeben sich schnell Koordinationsprobleme, da immer sehr viele Makros in jeweils einer Datei zusammengefasst sind und somit nur von jeweils einem Entwickler bearbeitet werden können. Viele Funktionalitäten in PP2000 lassen sich aber nur im Gesamtzusammenhang der Software testen und sind somit nur schlecht für einzelne Entwickler als Aufgabe zu isolieren. Mit steigender Anzahl von Entwicklern ist ein stark zunehmender Koordinierungsaufwand notwendig, um die Gesamtsoftware konsistent zu halten. Ein möglicher und auch beschrittener Weg, ist das Sammeln der bearbeiteten Makros jedes Entwicklers jeweils in einer separaten Datei, die in regelmäßigen Abständen in einer gemeinsamen Konferenz aller Entwickler in die zentral gesicherten Quellmakros eingepflegt werden. Wurde ein Makro von mehreren Entwicklern dabei gleichzeitig bearbeitet, so müssen diese gemeinsam eine Version des Makros erstellen, die alle notwendigen Anpassungen enthält. Nach der Zusammenführung aller Entwicklungsversionen, muss das neue Gesamtpaket noch getestet werden und anschließend die verschlüsselten Dateiversionen erstellt werden, damit immer eine aktuelle Kundenversion zur Verfügung steht.

Zusammengefasst lassen sich aus dem hier wiedergegebenen IST-Zustand die folgenden Probleme für die Entwickler und den Support ableiten:

- Der Quellcode ist überwiegend nicht kommentiert, eine anderweitige Dokumentation existiert nicht. Gleichzeitig verlangen die nachfolgenden neuen Versionen des zugrundeliegenden CAD-Systems ständig nach Anpassungen der Makroumgebung. Gleiches resultiert aus Forderungen von Kundenwünschen und aus Weiterentwicklungen der konkurrierenden Makropakete. Das Format bestimmter Konfigurationsdateien ist nicht dokumentiert, so dass an den entsprechenden Modulen keinerlei Veränderungen mehr vorgenommen werden können und für Anpassungen umständliche „Workarounds“ erstellt werden müssen.
- Nicht alle Makros, die das Gesamtpaket beinhaltet, sind auch in den unverschlüsselten Quellmakros enthalten. Dies bedeutet, dass die Quellen nicht vollständig sind und nur schwer nachvollzogen werden kann, welche Funktion die fehlenden Makros besitzen.

PP2000 lässt sich aber ohne die verschlüsselten Makros, von denen die Quellen nicht existieren, nicht starten.

- Für das Testen ist immer eine vollständige PP2000-Umgebung notwendig, dass heißt, es lassen sich die Funktion einzelner Module nicht unabhängig von den anderen testen. Bei auftretenden Problemen in einzelnen Funktionsbereichen, lassen sich die entsprechenden Codeabschnitte aufgrund mangelhafter Modularisierung des Systems nur schwer isolieren, um somit auf den Kern des Problems zu stoßen. Denn ein Fehler muss nicht immer ein Problem der PP2000-Software sein, sondern dieser kann auch ursächlich im ME10 begründet sein, z.B. durch die Einführung einer neuen ME10-Version.
- Manche Makros sind in PP2000 mehrfach vorhanden, so liegt z.B. das Makro **As_m_bom_pos_draw_box** in drei verschiedenen Dateien pp_bom_asc_new.mac, pp_kon_asc_new.mac und pp_sec_asc_new.mac in zwei unterschiedlichen Versionen vor, wobei nicht klar ist, welches davon die aktuellste und neueste Version ist (vgl. Abbildung 6.2), zumal alle Dateien den Zusatz „**new**“ enthalten. Es ist davon auszugehen, dass die Version in der zuletzt geladenen Makrodatei die aktuelle ist.
- Bei der Dateivorschau im PP2000-eigenen Dateimanager, gibt es bei der Hervorhebung und Identifizierung von Einzelteilen ein Problem, welches zum Absturz des Systems und somit zum vollständigen Datenverlust führen kann. Das manuelle Nachvollziehen der einzelnen Vorgänge im Makroablauf erbrachte jedoch niemals ein Problem und auch Ablaufprotokolle zeigten keine Fehler im Makroablauf. Aufgrund von Kundendruck entschloss man sich, die entsprechenden Codepassagen neu zu entwickeln und dabei eine etwas andere Vorgehensweise bei der Umsetzung zu wählen. Die Fehler traten jetzt allerdings immer noch auf, nur in anderen Zusammenhängen. Erst eine umständliche Isolierung der relevanten Codepassagen und das Entfernen aller für die Funktionalität überflüssigen Aufrufe ergab, dass der Code einwandfrei war und tatsächlich ein Fehler im Speichermanagement des ME10-Systems der Grund für die Abstürze war.

Die hier beschriebenen Probleme verdeutlichen noch einmal eindrucksvoll, dass für eine konsequente Weiterentwicklung von PP2000 zumindest eine Entwicklungsumgebung für den ME10-Makrocode notwendig ist, die dem Entwickler gewisse häufig durchgeführte Arbeits-

schritte, wie das Nachladen der Makros, einen automatischen Start der ME10-Trace-Funktion beim Testen der Makros und zumindest solche Arbeitserleichterungen wie das sogenannte Syntax-Highlighting zur Verfügung stellt, welches dem Programmierer die syntaktische Richtigkeit seines Codes anzeigt.

Des weiteren sollte die Entwicklungsumgebung auch notwendigerweise eine Quellcode-verwaltung, ein so genanntes Repository enthalten, welches die Versionsverwaltung übernimmt. Der größtenteils nicht kommentierte bzw. dokumentierte Code erfordert darüber hinaus, dass die Quellcodeverwaltung noch um eine Dokumentationsverwaltung ähnlich der eines PDM-Systems zu erweitern ist. Diese Überlegungen und die Tatsache, dass es auch in anderen Entwicklungsbereichen ähnliche Probleme gibt, verstärkten den Wunsch nach der Schaffung eines SLM-Systems, welches die Softwareentwicklung plattform- und bereichs-unabhängig unterstützt.

6.3 Struktur und Aufbau von PPNEU

Der gesamte Funktionsumfang der Vorgänger-Software PP2000 muss ohne jegliche Einschränkung erhalten bleiben. Die Neuentwicklung muss zu einem stabilen und fehlertolerantem System führen, welches insbesondere im Hinblick auf die Supportfähigkeit und die Weiterentwicklung deutlich bessere Eigenschaften aufweist als das Vorgänger-produkt. Die Benutzeroberfläche muss modularisiert aufgebaut werden, damit der nachträgliche Implementierungsaufwand für andere Betriebssysteme so gering wie möglich gestaltet werden kann. Neben diesen Anforderungen bezüglich der Funktionalität und der Benutzeroberfläche existieren noch die folgenden softwaretechnischen Anforderungen:

- Unterstützung der Betriebssysteme Windows 2000, Windows XP und UNIX,
- neben der deutschen Sprache müssen andere Sprachen integriert werden können, ohne dass die Quelltexte der Funktionen bearbeitet werden müssen,
- als Zielplattform ist die neueste Version von ME10 vorzusehen,
- eine modulartige Architektur muss einen strukturierten Aufbau gewährleisten,

- das neue Produkt muss sowohl die PE-Look-Oberläche als auch den Mixed-Look unterstützen,
- die nachträgliche Implementierung neuer Module muss möglich sein,
- die Benutzeroberfläche muss programmintern von der Funktionsebene getrennt sein und
- das Pflegeverhalten und der Support muss erheblich vereinfacht werden.

Die Realisierung der hier vorab genannten Forderungen verlangen einen strukturierten Aufbau des neuen ProfiPlus-Systems (**PPNEU**). Hierbei muss generell die Benutzeroberfläche getrennt von der ProfiPlus-Funktionalität implementiert werden. Das heißt, die ProfiPlus-Funktionen werden zukünftig aus der Benutzeroberfläche und die ME10-Funktionen aus der ProfiPlus-Funktionsebene aufgerufen. Diese Vorgehensweise gestattet es, andere Benutzeroberflächen zu konfigurieren, ohne dass unerwünschte Seiteneffekte in der Funktionsweise entstehen.

Für die Umsetzung dieser Forderungen wird hier eine 3-Ebenen-Architektur gewählt (vgl. Abbildung 6.5, linke Seite). Diese beinhaltet in der:

- Benutzerebene: die PPNEU-Benutzeroberfläche
- logischen Ebene: Trennung von Benutzer- und Datenebene
- Datenebene: die ME10-Funktionen.

Die PPNEU-Benutzeroberfläche ist jeweils abhängig von der verwendeten ME10-Version und muss sowohl für den Windows- als auch für den PE-Look getrennt definiert werden. Durch diese Vorgehensweise können die Definition der reinen Benutzeroberfläche an die Kunden weitergegeben werden, da dort keinerlei Logik integriert ist. Darüber hinaus kann der Kunde Erweiterungen an der Oberfläche vornehmen. Die Logik der Benutzeroberfläche stellt die Makros zur Verfügung, welche die Abarbeitung von Benutzerkommandos übernehmen. Diese logische Schicht kennt alle bereitgestellten Module und verknüpft diese mit der Oberfläche. Es befindet sich in dieser Schicht noch keinerlei Anwendungslogik. Herauszustellen ist, dass diese Schicht unabhängig von der jeweils verwendeten

Benutzeroberfläche eingesetzt werden kann. Die Weitergabe an den Kunden sollte nur in binärer Form erfolgen. Die Module der Benutzeroberfläche enthalten alle:

- in der Oberfläche sichtbaren PP-Elemente, wie Menüs, Dialoge, Tabellen etc.,
- verwendeten Textressourcen und Icons,
- Funktionalitäten zum konfigurationsabhängigen Einlesen von Textressourcen oder Icons und
- Funktionen zur Verarbeitung von Benutzereingaben.

Die PPNEU-Funktionsebene enthält die eigentliche ProfiPlus-Funktionalität gemäß PP2000 und gliedert sich in zwei Bereiche auf, einen Bereich in welchem die Funktionsmodule definiert sind und in einen Bereich, in dem innerhalb einer Toolbox sogenannte Hilfsmakros abgelegt sind. Diese logische Ebene enthält:

- alle zu realisierenden ProfiPlus Funktionalitäten,
- alle Standardfunktionalitäten von ProfiPlus, wie z.B. die Verarbeitung von logischen Tabellen (LTAB) und generalisierte Dialogfunktionen.

Die hier genannten LTAB's werden vorrangig zur Speicherung aller Konfigurations-einstellungen benutzt, um die Verwendung von globalen Variablen so weit wie möglich zu vermeiden..

Auf die Datenebene muss hier nicht besonders eingegangen werden, da es selbstverständlich ist, dass alle ME10-Funktionen von PPNEU aufrufbar sein müssen.

Wenn die schematische Modulstruktur (vgl. Abbildung 6.5) auf die Struktur des PPNEU in Verbindung mit dem CAD-System ME10 übertragen wird, so ergibt sich nunmehr für das System PPNEU eine Modul- und Dateistruktur (vgl. Abbildung 6.6).

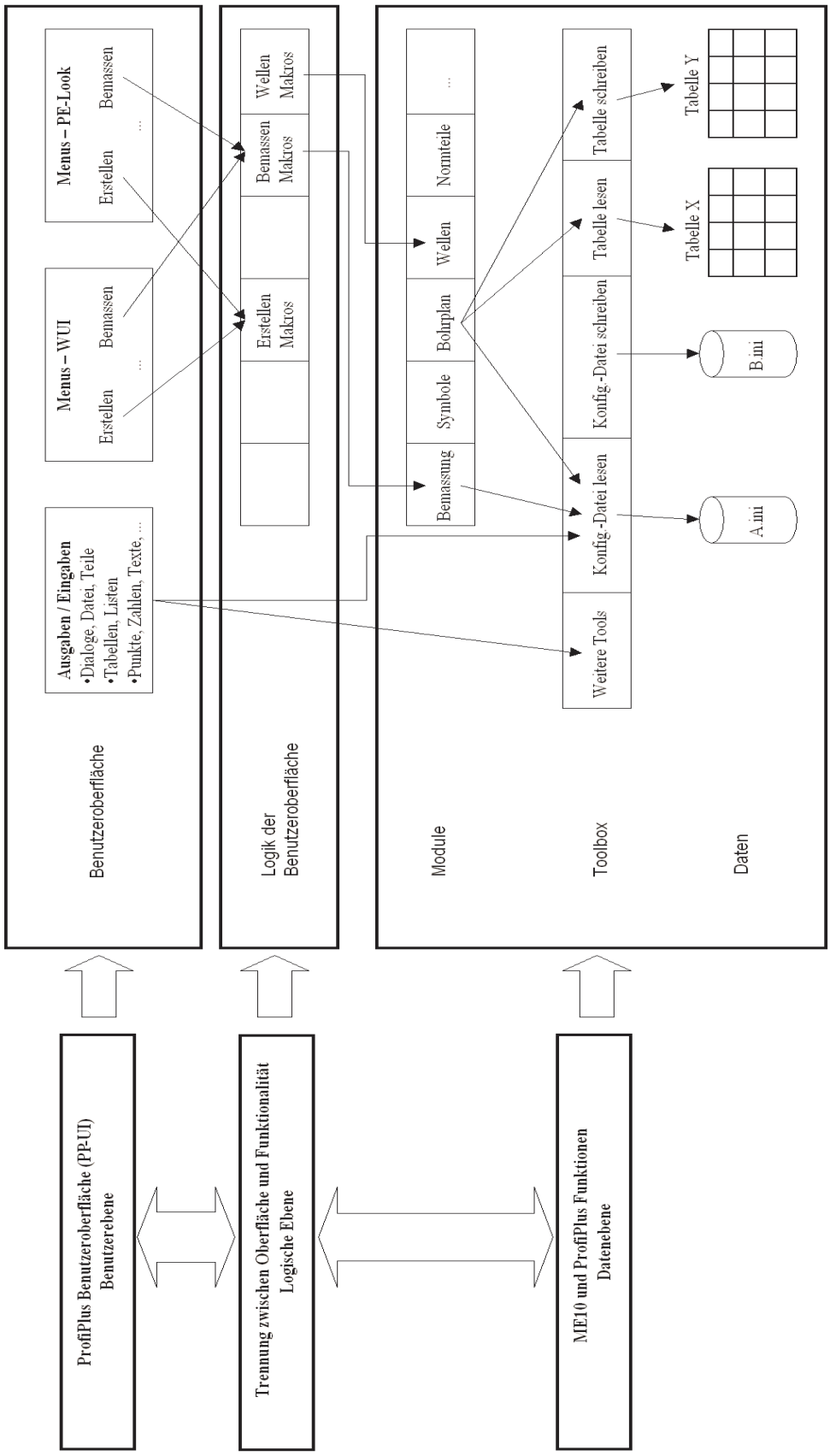


Abbildung 6.5: Schematische Modulstruktur von PPNeu

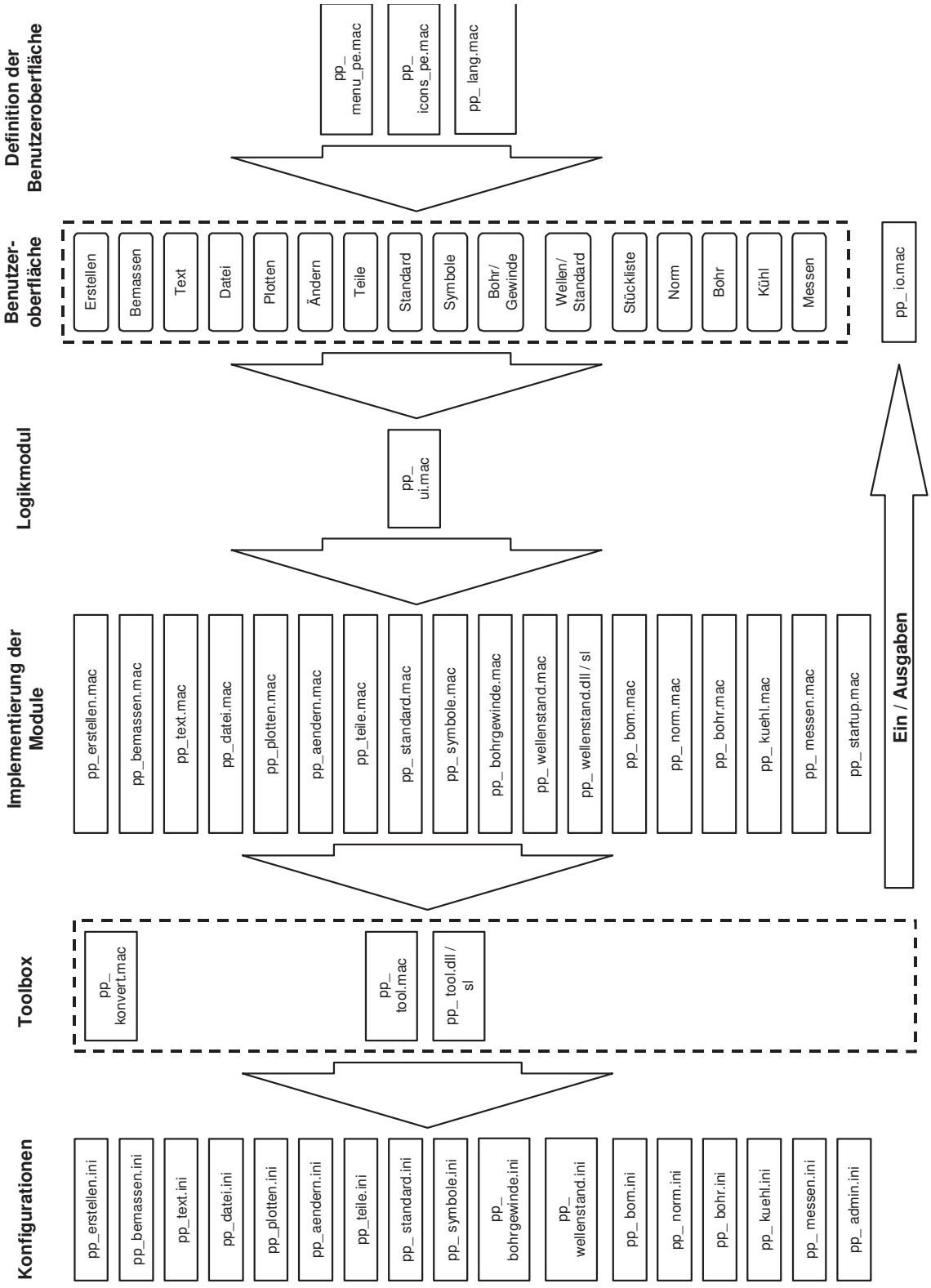


Abbildung 6.6: Übersicht der Module und Dateien

Ein Modul unter ProfiPlus ist prinzipiell die funktionale Einheit von Quelltexten, die einen speziellen, thematisch zusammenhängenden Teil der Software abbilden. Dies können sowohl ME10-Makros oder auch Bibliotheken sein, welche die Funktionalität durch Zugriff auf die Anwendungsschnittstelle von ME10 in C/C++ Quellcode abbilden. So enthält z.B. die Datei „erstellen.mac“ alle Makros, die zur Implementierung der im „Erstellen-Menu“ vorhandenen Funktionalität erforderlich sind oder die Bibliothek „pp_wellenstand.dll“ sämtliche Funktionalitäten zum vereinfachten Zeichnen von abgesetzten Wellen. Über die Dll’s werden im ME10 zusätzliche Schlüsselwörter zur Verfügung gestellt, die wie ME10-eigene Makrobefehle aufgerufen werden können. Innerhalb der Dll kann direkt auf die interne Datenstruktur der Geometrie zugegriffen und so Manipulationen daran durchgeführt werden, die mit den ME10-eigenen Makrobefehlen gar nicht oder nur umständlich zu realisieren sind. Ein Anwendungsfall hierfür ist die Funktionalität des Wellenstand-Moduls, die eine direkte Manipulation der Geometriedaten erfordert.

Die Definition der Benutzeroberfläche besteht aus den Moduldateien, welche die Aufteilung der Bildschirmmenus, die darzustellenden grafischen Symbole und die anzuzeigenden Texte in den jeweiligen Sprachen enthalten. Die Benutzeroberfläche enthält zu jeder Schalterfläche der Menus die aufzurufende PPNEU- oder ME10-Funktionalität. Welche Funktionalität aufgerufen ist, wird im Logikmodul spezifiziert, ob eine PPNeu-Funktionalität oder direkt eine ME10-Funktionalität aufgerufen wird. Für die Ausführung der PPNEU-Funktionalitäten werden bestimmte, konfigurierbare Daten benötigt, die in den Konfigurationsdateien enthalten sind. Der Zugriff auf diese Daten erfolgt über die Module der Toolbox.

Die Abbildung der Modul-Struktur erfolgt eins zu ein auch in der Dateistruktur. Die Funktionalitäten der Module, die in den einzelnen Makros abgebildet werden, sind Modul für Modul jeweils in einer einzelnen Datei zusammengefasst (vgl. Abbildung 6.6), die in der Abbildung gemäß ihrer Aufrufreigenfolge von rechts nach links dargestellt sind. Die Oberflächenmakros rufen die Makros der Benutzeroberfläche auf, die Makros der Benutzeroberfläche die des Logikmoduls, usw. Diese Modul-Makrodateien werden wie beim PP2000 in einem einzelnen Unterverzeichnis „Makros“ unterhalb des ProfiPlus-Installations-verzeichnisses abgelegt, die Dll’s entsprechend in einem Unterverzeichnis „dll“ und die Konfigurationsdateien in einem Unterverzeichnis „konfig“.

6.3.1 Dateiaufbau

Während der Erstellung von PPNEU werden verschiedene Dateiformate verwendet, welche alle vom SLM-System verwaltet und unterschieden werden müssen. Dies ist insbesondere wichtig bei der Anlage von Metadokumenten (vgl. Kap. 6.3.2) zu Dokumentationszwecken und für die Ausführung dateitypspezifischer Skripte (vgl. Kap. 5.3.9) durch das SLM-System bei definierten Vorgängen, wie etwa einem Ein- oder Auscheckvorgang.

Für die Klassifizierung innerhalb des SLM-Systems und für die automatische Anlage von Metadokumenten (vgl. Kap. 5.3.4) werden die einzelnen Dateitypen innerhalb von PPNEU wie folgt festgelegt:

Dateityp	**Beschreibung**	**Dateiendung**
ME10-Moduldatei	Enthält alle Makros eines Moduls (Bohrbilder, Welle)	*.mac
ME10-Makrodatei	Datei, aus denen sich die Moduldateien zusammensetzen und die nur ein einzelnes Makro enthalten, Sie wird durch das SLM-System über Skripte aus den Moduldateien extrahiert. (vgl. Kap. 6.5.3)	*.mac
C-Quellcodedatei	Für alle Module, die in Bibliotheken überführt werden.	*.c / *.cpp
C-Headerdatei	Enthält Deklarationen für die C-Quellcode-Dateien	*.h
Script-Datei	Diese Ausführungsdatei wird vom Betriebssystem interpretiert und hauptsächlich zur automatischen Konfiguration unter Unix verwendet	*.sh
Konfigurationsdatei	Enthält die Konfigurationseinstellungen der Module	*.ini
Ausführungsdatei	Maschinencode-Datei, die direkt vom Prozessor ausgeführt wird	*.exe
Bibliotheksdatei	Maschinencode-Datei, die nur im Zusammenhang mit einer Ausführungsdatei ausgeführt werden kann	*.dll (Windows); *.sl (Unix)
Geometriedatei	Enthält die Symbole für die ProfiPlus-Oberfläche	*.mac
Messagedatei	Enthält die Ein- und Ausgabetexte des Systems in verschiedenen Sprachen	*.mac
Dokumentationsdatei	Enthält die Projektdokumentationen enthält, die nicht mit einer anderen Datei (Metadatei) verknüpft sind	*.doc; *.prj; ...
ME10-Zeichnungs-Datei	Datei, die ME10-Zeichnungsdaten enthält, z.B. für vorkonfigurierte Zeichnungsrahmen	*.mi

Tabelle 6.1: Übersicht über die Dateitypen und deren Endungen

Die Geometrie und Message-Dateien sind ebenfalls Dateien im ME10-Makroformat, sie enthalten jedoch nur Wertzuweisungen für Variablen. Die Geometriedarstellungen in den Menus von ME10 erfolgen über einen unübersichtlichen Textcode und sind nur mühsam manuell zu generieren.

Dokumentationsdateien können im Prinzip jede Dateiendung haben, in Abhängigkeit von dem zur Dokumentation verwendeten Programm. Wurde für die Dokumentation MS-Word verwendet, hat die Datei also die Endung *.doc.

6.3.2 Inhalte der Metadokumente

Als Metadokumente im SLM-System werden Dokumente bezeichnet, die Informationen über andere Dokumente enthalten. Sie werden im Rahmen des PPNEU-Konzeptes hauptsächlich für die Dokumentation der Entwicklungsschritte durch die Entwickler selber verwendet.

Metadokumente enthalten also zusätzliche Informationen über die einzelnen erstellten Dateien und dienen in erster Linie der Dokumentation des Projektes, der Module und der Makrodateien. Sie werden, wie in Kapitel 5.3.4 dargestellt, immer bei der Anlage eines neuen Dokumentes im SLM-System oder bei einem bestimmten Statusübergang, entsprechend der Konfiguration des Projektes im SLM-System erstellt. Die Neuanlage eines Dokumentes findet sowohl bei der eigentlichen Neuerstellung eines Dokumentes, als auch bei jedem Check-In Vorgang statt, da die alten Dokumente im SLM-System verbleiben. Im Falle eines Check-In muss eine Änderungsverfolgung vorgenommen werden, so dass klar wird, was sich von der aktuellen Version zur Vorgängerversion geändert hat.

Für die Dokumentation der ME10-Modul- und Makrodateien genügen einfache Textdateien für die Dokumentation. Dies bietet gleichzeitig den Vorteil, dass der gleiche Editor zur Makroerstellung und für die Erstellung der Dokumentation verwendet werden kann. Über ein Skript kann dann auf einfache Weise eine Gesamtdokumentation des Projektes erstellt werden. In den Dateien von den Entwicklern Informationen über Art und Zweck der Datei eingetragen, die Parameter, Anwendungsbeispiele usw. (vgl. Abbildung 6.7 und Abbildung 6.8).

```
Modul-Name:         Erstellen

Zweck:              Erstellen-Funktionalität im ProfiPlus

Herkunft:           pp_erstellen.mac

Dateien:            pp_erstellen.ini, pp_erstellen.mes

Referenz:           pp_startup.mac

Beschreibung:       Enthält alle Makros, die zum Modul Erstellen
                    gehören

Ersteller           Jens Stolpmann
```

Abbildung 6.7: Dokumentationsdatei (Neuanlage Modul-Datei)

```
Makro-Name:         pp_read_ini

Zweck:              Liest die jeweilige .ini-Datei in die Tabelle
                    pp_temp_ini_ltab

Herkunft:           pp_ini.mac

Rueckgabe:          -

Parameter:          pp_ini_pfad: enthält die Pfadangabe für die zu lesende
                    .ini-Datei

Dateien:            -

Nachrichten:        -

Seiteneffekt:       Überschreibt den Inhalt der Tabelle  pp_temp_ini_ltab

Beispiel:           pp_read_ini "c:/opt/Cocreate/pp2002/pp_erstellen.ini"

Ersteller           Jens Stolpmann

Datum               01.01.2003
```

Abbildung 6.8: Dokumentationsdatei (Neuanlage ME10-Makrodatei)

Die Informationen über Ersteller und Datum werden bei der Anlage des Dokumentes automatisch den Metainformationen des Dokumentes entnommen und in die Datei eingetragen, so dass bei der Zusammenstellung der Informationen und deren Ausdrucken diese nicht mehr dem SLM-System entnommen werden muss.

6.4 Programmsteuerung von PPNEU

6.4.1 Startvorgang

Wie bereits ausführlich dargestellt, ist der Startvorgang der alten Version PP2000 sehr stark verschachtelt. Bereits kleine Abweichungen bzw. Änderungen führen dazu, dass das PP2000 nicht mehr korrekt startet. Aufgrund des Startverhaltens von ME10 ist es in einem solchen Fall extrem schwierig festzustellen, an welcher Stelle der Startvorgang gescheitert ist. Dies führt bei der Installation zu umfangreichen Such- und Testvorgängen. Aus diesem Grund wird in PPNEU eine neue Startprozedur vorgesehen. Hier wird nur noch eine Startprozedur mit geringer Verschachtelungstiefe implementiert. Außerdem wird ein Mechanismus entwickelt, der es ermöglicht vor dem Einladen einer Makrodatei festzustellen, ob diese Datei auch vorhanden ist.

Dazu ist die Erstellung einer AI-Funktion erforderlich, welche das Vorhandensein einer Datei überprüft, und für den Fall, dass die Datei nicht geladen werden kann, den Startvorgang von PPNEU mit einer eindeutigen Fehlermeldung abbricht. Diese Forderung bedingt für den Systemstart von PPNEU einen generellen Ablauf (vgl. Abbildung 6.9). Dabei werden zunächst die erforderlichen AI-Module geladen, da diese bereits Funktionen enthalten, die den ordnungsgemäßen Start von PPNEU gewährleisten. Im Anschluss wird die aktuelle Konfiguration aus den Ini-Dateien eingelesen. Diese Konfiguration definiert, welche Makros (z.B. Sprachunterstützung oder Benutzeroberfläche) im nächsten Schritt eingeladen werden.

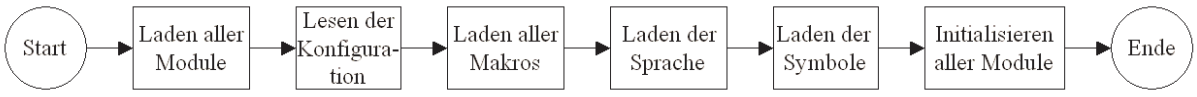


Abbildung 6.9: Startvorgang für PPNEU

6.4.2 Ignorieren fehlerhafter Makros

Während der langen Zeit, in der PP2000 bereits im produktiven Einsatz ist, hat sich ein besonderes Systemverhalten als besonders schwerwiegender Mangel herausgestellt. Dieser konnte jedoch auf Grund der bereits beschriebenen Schwachstellen im Konzept von PP2000 nie behoben werden. Der Umstand, dass im Rahmen der Ausführung eines Makros oftmals die Systemumgebung von ME10 temporär verändert wird führt zu undefinierten Systemzuständen, falls ein solches Makro, auf Grund eines Fehlers oder durch Abbruch durch den Benutzer, beendet wird, bevor der ursprüngliche Systemzustand von dem Makro wieder hergestellt werden konnte. Im Zuge der Entwicklung von PPNEU wird diese Schwachstelle wie folgt behoben.

Bei der Generierung der Makros sind jeweils Mechanismen zur Programmsteuerung vorzusehen, damit sichergestellt wird, dass im Falle eines Fehlers während der Abarbeitung eines Makros, der vor dem Aufruf gültige Systemzustand wiederhergestellt wird. Dazu wird als Teil der Logik-Ebene eine so genannte AI-Funktion implementiert, die es ermöglicht ein generelles Reset-Makro zu definieren. Dieses Reset-Makro enthält die jeweils notwendigen Befehle zur Wiederherstellung des ursprünglichen Systemzustandes (z.B. die voreingestellte Linienart oder Farbe) und zwar sowohl für ME10 als auch für PPNEU (vgl. Abbildung 6.10).

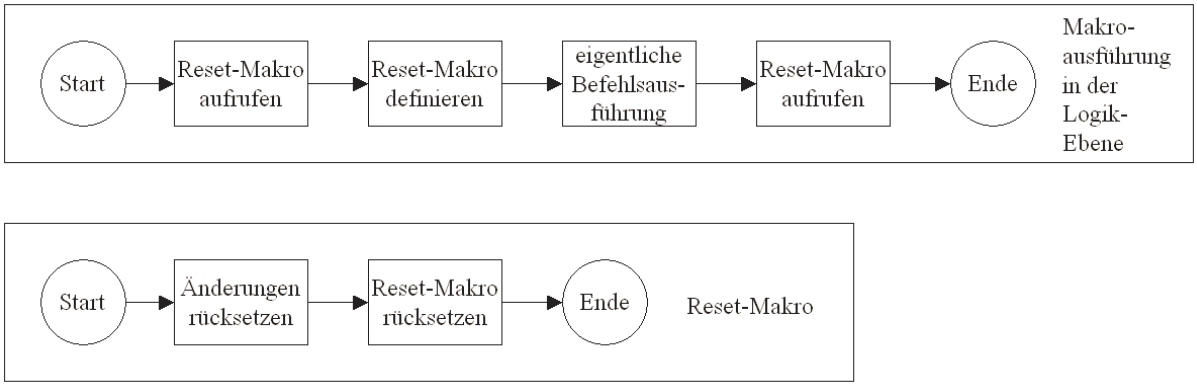


Abbildung 6.10: Ausführung von Makros

Die Ausführung eines Makros geschieht dann immer nach folgendem Muster. In dem Befehlsmakro der Logik-Ebene wird vor der eigentlichen Makroausführung das Reset-Makro aufgerufen, um die Systemeinstellung nach einem eventuellen Fehler in einem zuvor aufgerufenen Makro wieder herzustellen. Wenn in dem Befehlsmakro Änderungen an Systemeinstellungen vorgenommen werden, müssen die Befehle zur Wiederherstellung der aktuell gültigen Systemeinstellungen jetzt in einem neuen Reset-Makro definiert werden. Dies geschieht mit Hilfe einer AI-Funktion. Dann erfolgt die eigentliche Befehlsverarbeitung. Vor Beenden des Makros werden die Systemeinstellungen durch Aufrufen des Reset-Makros wiederhergestellt.

Dazu wird eine Cancel-Funktion implementiert, die das Abbrechen eines PPNEU-Befehles ermöglicht. Betätigt der Benutzer innerhalb einer PPNEU-Wiederholfunktion, wie z.B. „Linie Polygon“ den Abbrechen-Schalter, so wird die Funktion abgebrochen. Die Funktionsweise der Cancel-Funktion entspricht der Standardverarbeitung von ME10, das heißt, bereits eingegebene Zeichen werden gelöscht und evtl. reservierter Speicher wird wieder freigegeben.

Anders als in der Programmiersprache C können in der ME10-Makrosprache keine Returnwerte für Makros verwendet werden. Returnwerte werden in der Programmierung oft als Statuswerte zurückgegeben, die Informationen darüber enthalten, ob eine Funktion fehlerfrei ausgeführt wurde. So kann in einer Funktion nach dem Aufruf einer untergeordneten Funktion der Returnwert dieser Funktion abgeprüft werden. Auf diese Art kann auf einen Fehler in einer untergeordneten Funktion individuell reagiert werden.

Um einen solchen Mechanismus auch in den PPNEU-Makros zu nutzen, wird folgende Richtlinie für die Programmierung von ME10-Makros festgelegt:

- Es wird eine globale Variable für einen Returnwert definiert.
- Diese Variable hat den Namen: **As_vg_ret**
- Diese Variable wird mit dem Wert 0 vorbelegt.
- Der Wert 0 entspricht dem Rückgabewert „Kein Fehler“
- Für differenzierte Angaben über die Art eines aufgetretenen Fehlers werden ganzzahlige Fehlerwerte verwendet, wie z.B.:
    - 1 := genereller Fehler
    - 2 := Datei nicht gefunden

etc.

6.4.3 Konfigurationsmanagement

In dem System PP2000 sind die Konfigurationseinstellungen in einer Vielzahl von Dateien abgelegt. Historisch bedingt, ist heute nicht mehr klar, welche Einstellung in welcher Datei zu welchem Zeitpunkt von PP2000 verwendet wird. Aus diesem Grund muss das System PPNEU zumindest in einer Übergangsphase auch alte Dateien aus dem System PP2000 einlesen können. Deshalb muss im Rahmen des Redesigns für PPNEU ein neues Konfigurationsmanagement implementiert werden.

Alle Konfigurationseinstellungen werden in ASCII-Dateien vorgenommen, deren Format den von Windows bekannten INI-Dateien entsprechen. Auf diese Weise wird sichergestellt, dass die Konfiguration von PPNEU in einer transparenten Art und Weise auch durch direktes Editieren dieser Dateien vorgenommen werden kann. Der Aufbau einer Konfigurationsdatei wird in Abbildung 6.11 gezeigt:

```
[Admin Linie]
LINETYPE=DASHED
COLOR=CYAN
LINEWIDTH=0
...

[Admin Text]
Text_Color=WHITE
TEXT_ADJUST=1
TEXT_ANGLE=0
...
```

Abbildung 6.11: Ini-Dateien von PPNEU (Admin.ini)

Die Konfigurationsdatei besteht aus verschiedenen Sektionen. Eine Sektion wird durch den Sektionsnamen (z.B. [Admin Linie]) gekennzeichnet, eingeklammert durch „[]“. Eine Sektion gilt solange bis eine neue Sektion (hier: [Admin Text]) definiert wird. Innerhalb einer Sektion werden die Voreinstellungen nach dem Muster

<Name> = <Wert>

vorgenommen. Dabei kann für <Wert> sowohl eine Zahl als auch eine Zeichenfolge verwendet werden. Für die Handhabung solcher Ini-Dateien sind PPNEU-Makros zu implementieren, die es erlauben, Werte aus der Datei zu lesen bzw. in die Datei zu schreiben. Diese Makros gehören zu der PPNEU-Toolbox (vgl. Abbildung 6.5 und Abbildung 6.6).

Unabhängig von diesen Konfigurationsdateien wird ein Konfigurationsmodul implementiert, welches die Einstellung von PPNEU über die Benutzeroberfläche gestattet. Dieses Modul ist mit der PP2000-Funktion „Admin“ vergleichbar. Zum Importieren einer Konfiguration einer PP2000-Version ist die Erstellung eines Import-Programmes erforderlich. Dieses Programm liest mit Hilfe von PP2000-Funktionen die Konfigurationseinstellungen der gewünschten PP2000-Version ein und aktualisiert die INI-Dateien in PPNEU entsprechend.

6.4.4 Möglichkeiten der Lizensierung

Der Mechanismus zur Lizensierung soll sicherstellen, dass PPNEU ausschließlich nur autorisiert verwendet werden kann. Hier kann der von CoCreate entwickelte License Manager (MELS) berücksichtigt werden, der auch für die Basissoftware ME10 genutzt wird. Bezüglich der Einsatzfähigkeit muss in einer Konfigurationsdatei des License Managers ein Passwort eingetragen werden. Dieses Passwort erhält der Kunde vom Anbieter der Software und es legt fest, welche Softwaremodule in welchem Umfang genutzt werden können. Die zu lizensierende Software, in diesem Falle PPNEU, richtet eine Anfrage an den License Manager und erhält von diesem eine Antwort in der Form „Lizenz gewährt“ oder „Lizenz nicht gewährt“, falls kein Passwort gefunden wurde oder bereits alle verfügbaren Lizenzen in Gebrauch sind.

Für das Produkt PPNEU wird auch in Hinblick auf die Lizensierung ein modulartiger Aufbau realisiert. Für einen bestimmten Umfang der gesamten Funktionalitäten wird ein Grundpaket PPNEU-Basis definiert. Darüber hinaus können weitere Funktionalitäten als Zusatzmodule, z.B. das Modul ZVS (Zeichnungsverwaltung) dem Anwender angeboten werden. Diese Zusatzmodule müssen über einen eigenen Lizensierungscode verfügen.

Das Abprüfen der Lizenz findet für das Grundpaket PPNEU-Basis bei jedem Systemstart statt. Die Lizenzabfrage erfolgt während der Initialisierung in einer AI-Routine. Dadurch wird sichergestellt, dass die Lizenzabfrage nicht mit einfachen Mitteln umgangen werden kann, wie dies bei einem Aufruf aus der Makroebene möglich wäre.

Wenn keine Lizenz für den Betrieb von PPNEU gewährt wird, bleiben die AI-Module ungeladen und es findet keine Initialisierung statt. Somit ist ein Betrieb von PPNEU nicht möglich. Durch das Fehlen der Initialisierungseinstellungen sind auch die Makro-Module nicht betriebsbereit. Es wird eine Fehlermeldung ausgegeben und der Systemstart wird abgebrochen.

Für die Prüfung der Lizenzen für die Zusatzmodule wird das folgende Verfahren angewendet. Vor der Abfrage der Lizenz prüft die Initialisierungsroutine, ob das CAD-System ME10 im Testmodus läuft. Ist der Testmodus aktiv, so findet keine Lizenzabfrage statt. Es erfolgt eine entsprechende Meldung an den Benutzer und PPNEU startet ebenfalls im Demo-Modus. Dieser Modus ist gegenüber der regulären Betriebsart nicht eingeschränkt. Es werden lediglich die Einschränkungen des Testmodus von ME10 ausgenutzt. Intern wird jedoch in der Konfiguration ein Schalter auf DEMO-Modus gestellt.

Durch die Verwendung eines Demo-Modus wird Interessenten die Möglichkeit gegeben, das Softwaresystem PPNEU zu testen, bevor eine Kaufentscheidung getroffen wird. In der Dokumentation ist die Möglichkeit des Testbetriebs deutlich zu erwähnen. Die Lizensierung muss sicherstellen, dass beim Starten einer zweiten PPNEU-Instanz keine zusätzliche Lizenz angefordert wird.

6.4.5 Verbesserungen der Support-Möglichkeiten

Als Maßnahme zur Verbesserung des Pflegeverhaltens wird eine Trace-Funktionalität implementiert. In der Ini-Datei **Admin** wird der Schlüssel „TRACE_LEVEL“ eingeführt. Dieser hat den Defaultwert „0“. Wird hier eine „1“ eingetragen, so läuft PPNEU im Trace-Modus und schreibt alle Systeminformationen in eine Log-Datei, deren Ansicht für den Support bei der Fehlersuche eine wesentliche Arbeitserleichterung darstellt.

Zur Umsetzung wird sowohl für die AI-Module als auch für die Makro-Module eine Ausgabefunktion CT_TRACE mit einem Parameter vom Typ Zeichenkette definiert. Diese Funktion prüft jeweils den eingestellten Trace-Level. Bei Trace-Level 0 wird die Funktion sofort beendet, ansonsten erfolgt die Ausgabe der Zeichenkette in die Log-Datei. Die Trace-Funktion ist zu Beginn und zum Ende einer jeden Funktion, bzw. eines jeden Makros aufzurufen sowie an besonderen Stellen des Quelltextes. Beispielhaft stellt der folgende Auszug die Trace-Funktion für die AI-Module dar (vgl. Abbildung 6.12):

```
void CT_TRACE (char *a)
{
	char sDebug[BUFSIZE];
	if ( ct_trace_level == 0 )
		return;
	else if ( ct_trace_level == 1 )
	{
		FILE *Fp = 0;
		Fp = fopen ( gTraceFile, "a" );
		if ( ! Fp )
		{
			me_display_prompt ( "TRACE FILE NOT FOUND" );
			return;
		}
		sprintf ( sDebug, "PP2002:  %s\n",  a );
		fprintf ( Fp, sDebug );
		fclose ( Fp );
	}
	else if ( ct_trace_level == 2 )
	{
		sprintf ( sDebug, "PP2002:  %s",  a );
		me_display_prompt ( sDebug );
	}
	return;
}
```

Abbildung 6.12: Trace Funktion für AI-Module (C-Quelltext)

Um das Auffinden eines Fehlers zu erleichtern, müssen alle Ausgabefunktionen ihre Informationen sowohl in die Bildschirmausgabe als auch in die Log-Datei schreiben.

Ein weiterer wichtiger Punkt für die Supportunterstützung ist die Erstellung einer geeigneten Dokumentation. Die Dokumentation in PPNEU erfolgt, wie bei ME10 im HTML-Format. Da im Rahmen des Redesigns der generelle Aufbau von ProfiPlus komplett neu erstellt wird, was besonders für die Konfiguration des Systems gilt, muss die Systemdokumentation

entsprechend der neuen Architektur von PPNEU neu erstellt werden. Hierbei ist zwischen den Begriffen Textressourcen und Quelltext zu unterscheiden. Zu den so genannten Textressourcen gehören beispielsweise:

- Fehler- oder Statusmeldungen
- Dialogüberschriften
- Menübeschriftungen
- Dialogvorgaben wie JA/NEIN oder OK/ABBRECHEN

Um eine Unterstützung der Mehrsprachigkeit des Produktes zu gewährleisten, ist eine strikte Trennung dieser Textressourcen vom Quelltext des Systems erforderlich. Das bedeutet, dass im Quelltext an keiner Stelle direkt Texte in irgendeiner Form ausgegeben werden dürfen. Alle Textressourcen werden in einer zentralen Datei definiert. Diese Datei enthält ausschließlich die Definition von Texten als ME10-Makros (vgl. Abbildung 6.13). Im Quelltext erfolgt lediglich eine Verwendung dieser Makros.

```
define As_m_lang_mac
     define Txt1    'OK'
     define Txt2    'Fehler"
     define Txt3    'Bitte Punkt identifizieren'
     define Txt4    'Kunde – PPNEU'
end_define
```

Abbildung 6.13: Auszug der Datei pp_lang.mac

Bei der Umschaltung auf eine andere Sprache muss nun lediglich eine andere Version dieser Textdatei geladen werden und PPNEU kann mit dieser Sprache betrieben werden, ohne dass Änderungen am Quelltext notwendig sind. Die Verwendung von Textressourcen wird an einem Beispiel-Makro erläutert (vgl Abbildung 6.13). Die zugehörige Datei wird beim Systemstart eingeladen und das Makro **As_m_lang_mac** wird ausgeführt, so dass die

definierten Textressourcen verfügbar sind. Derartige Textressourcen werden in allen PPNEU-Modulen verwendet, wie beispielsweise bei der Eingabeaufforderung:

6.5 Integration der Entwicklungsumgebung für PPNEU

Die Entwicklungsumgebung für die Generierung der ME10-Makros besteht analog zur Abbildung 5.1 aus drei Komponenten, dem SLM-System mit dem Client, einem geeigneten Editor und dem CAD-System ME10 als Code-Ausführungssoftware. Der Editor muss folgende Anforderungen erfüllen:

- Schnittstelle zur Fernsteuerung von Lade- und Speichervorgängen
- Bearbeitung mehrerer Dateien gleichzeitig
- Konfigurierbare Syntaxhervorhebung
- Einbindung von Tools in ein Custom Menu (zur Steuerung des Client)

Für dieses Projekt wird als Editor das Programm „Ultra-Edit“ ausgewählt (vgl. Abbildung 6.14), da dieser Texteditor speziell für die Anforderungen von Programmierern ausgelegt wurde. Zusätzlich zu den oben genannten, erforderlichen Funktionalitäten, bietet er noch viele weitere Funktionalitäten an, welche die Arbeit des Entwicklers erheblich vereinfachen, wie z.B. :

- sehr komfortable „Suchen- und Ersetzen“-Funktionen,
- eine Makroprogrammiersprache,
- die Möglichkeit Funktionslisten in einer oder mehreren Dateien anzeigen zu lassen,
- eine sehr gute automatische Formatierungsunterstützung, die einen gut lesbaren Quellcode erzeugt.

Eine weitere positive Eigenschaft dieses Editors ist, dass er die geöffneten Dateien immer überwacht und falls diese von dritter Seite verändert werden, den Benutzer augenblicklich darüber informiert. Dies ist sehr nützlich, wenn z.B. nach einem Testlauf in ME10 die Debug-Protokolldatei verändert wurde. Sie wird dann auf Nachfrage beim Benutzer automatisch geladen und angezeigt.

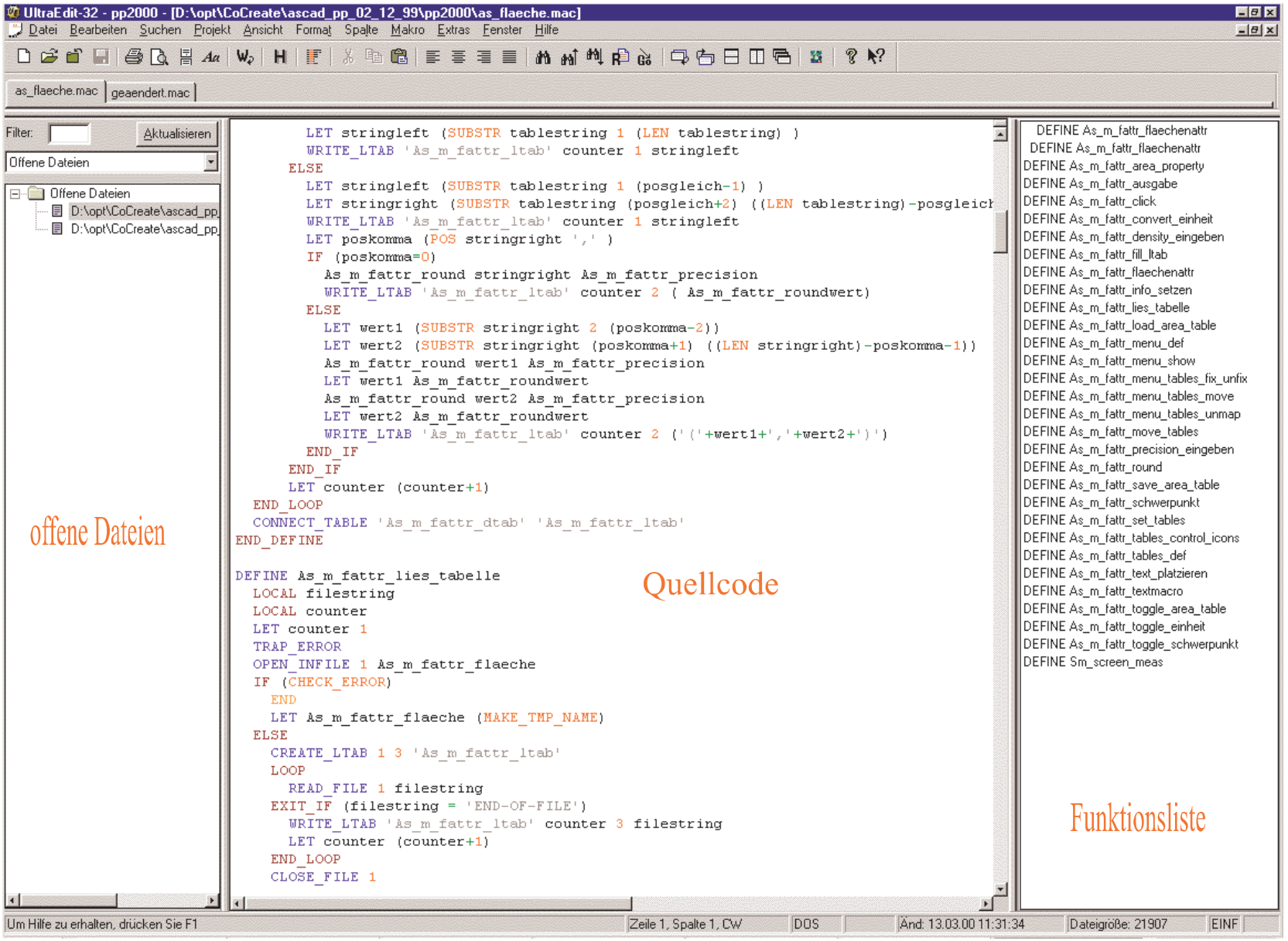


Abbildung 6.14: Ultra-Edit

Damit während der Entwicklungstätigkeit die Möglichkeit zum Testen der gerade vorgenommenen Änderungen an einer Makrodatei gegeben ist, muss die Ausführungssoftware, also ME10, in die Entwicklungsumgebung integriert sein. Diese Integration erfordert die folgenden Funktionen:

- Ferngesteuertes Laden einer Makrodatei
- Ferngesteuertes Ausführen einer Makrodatei
- Ferngesteuerter Start eines Debug-Protokolls, welches in einer externen Datei abgelegt wird.

Die Implementierung dieser Funktionen verwendet die Schnittstelle zu ME10, wie in Kapitel 6.5.2 beschrieben.

6.5.1 Schnittstelle zum Editor

Der Editor Ultra-Edit verfügt über zwei Möglichkeiten, mit anderen Programmen zu kommunizieren. Über das „Extras“-Menu lassen sich Aufrufe von externen Programmen konfigurieren (vgl. Abbildung 6.15).

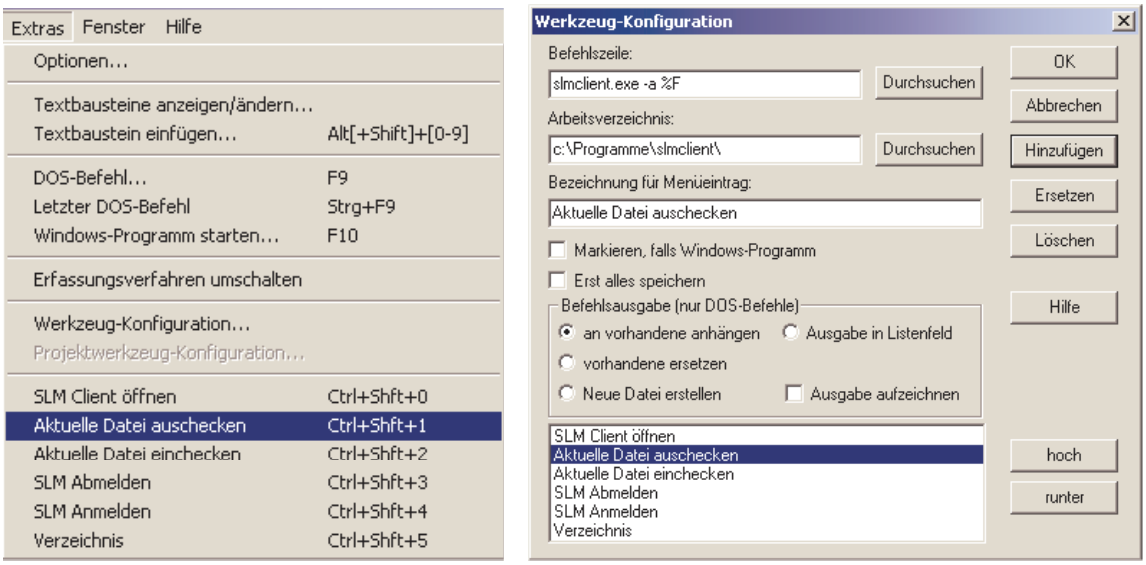


Abbildung 6.15: Werkzeug-Konfiguration in Ultra-Edit

Durch den Aufruf der SLM-Client-Ausführungsdatei mit den entsprechenden Optionen kann der Editor gesteuert werden. Mit der aktuellen Datei ist hierbei jeweils diejenige Datei in Ultra Edit gemeint, die gerade den Eingabefokus besitzt. Auf diese Weise können jeweils nur diejenigen Dateien identifiziert werden, die bereits in dem Editor geöffnet sind. Ein Zugriff von anderen Programmen kann mit Hilfe der DDE-Schnittstelle von Ultra-Edit realisiert werden. Hierbei agiert Ultra-Edit als DDE-Server, welcher eine Kommunikation über das Topic „System“ erlaubt. Der Aufbau einer Kommunikation zwischen dem SLM und dem Ultra-Edit geschieht folgendermaßen:

1. Es wird über die Betriebsystem-DDE-Funktion nach dem Programm „UEdit“ gesucht. Läuft kein Programm, wird Ultra-Edit gestartet.
2. An Ultra-Edit wird eine Anfrage gestellt, das Thema „System“ zur Verfügung zu stellen.
3. Es wird ein dem Thema „System“ zugeordneter Aufrufstring an Ultra-Edit geschickt. (z.B. [open("Dateiname" /M,E,1="vollständiger Pfad der Makrodatei/Makroname")])

Der String aus dem Beispiel öffnet eine Ultra-Edit-Makrodatei und führt ein Makro aus dieser Datei aus. Dies kann z.B. dazu verwendet werden, sofort gewünschte Änderungen an der Datei durchzuführen. Die Funktionen, die das Thema „System“ zur Verfügung stellt, sind die folgenden:

- [open("Dateiname")] Öffnet die angegebene Datei.
- [save()] Speichert das aktive Dokument.
- [print("Dateiname")] Druckt die angegebene Datei.
- [printto("Dateiname","Druckername","Druckertreiber","Port")]
  Druckt die angegebene Datei aus. Der zu verwendende Drucker wird durch den Druckernamen, den Druckertreiber und den Anschluss (Port) festgelegt..

Für die Integration werden allerdings nur die beiden ersten Punkte benötigt. So kann der open-Befehl für die Speicherung einer speziellen Datei benutzt werden, auch wenn dies nicht unbedingt die aktuelle Datei ist. Dies ist deshalb möglich, da nach dem Aufruf des open-Befehls die angegebene Datei immer zur aktiven Datei erklärt wird, auch wenn der Befehl für

eine Datei abgesetzt wird, die bereits im Editor geöffnet ist. Ein anschließender Save-Befehl führt also zu keinem Fehler.

6.5.2 Schnittstelle zu ME10

Das CAD-Programm ME10 bietet zwei Schnittstellen, mit denen es mit anderen Programmen kommunizieren kann. Zum ersten hat ME10 eine API, die es erlaubt, über die Einbindung einer DLL in ME10 auf die internen Datenstrukturen der Geometrie zuzugreifen. Des weiteren ist über die API die Erzeugung neuer Makrobefehle möglich, die dann in Makro-Quelltexten verwendet werden können Gleiches gilt für die Erzeugung von Makros. Weiterhin ist das Laden und Speichern von Dateien möglich, so wie die Interaktion mit dem Benutzer. Leider ist es über die API nicht möglich, Makro-Befehle in ME10 direkt auszuführen, so dass auch das Starten eines Makros und eines Debug-Protokolls nicht ausgeführt werden kann.

Daneben verfügt ME10, genau wie Ultra-Edit, über eine DDE-Schnittstelle, über die andere Programme mit ME10 kommunizieren können. Im Falle von ME10 sind die für den Aufbau einer DDE-Verbindung notwendigen Parameter „ME10F“ als Applikationsname und „GENERAL“ als Topic vorgesehen. Nach dem Aufbau der DDE-Verbindung zu ME10 kann nun als Zeichenkette jeder gültige ME10-Makro-Befehl übergeben werden, der dann unmittelbar ausgeführt wird. Wie bereits oben beschrieben, ist das ME10 in Zusammenhang mit der Softwareentwicklungsumgebung als passives Element zu sehen. Die Befehle, die zum Laden, Starten und Testen eines speziellen Makros über DDE an ME10 übergeben werden, sind im Folgenden wiedergegeben:

```
INPUT '<Pfad/MakroDateiName>'
TRACE DEL_OLD '<Pfad/TraceDateiName>'
<Makroname>
TRACE OFF
```

Unmittelbar nach dem “TRACE OFF”-Befehl, der die Protokolldatei schließt, wird das erstellte Debug-Protokoll automatisch vom Editor geladen und dem Benutzer angezeigt.

6.5.3 Ein- und Auschecken einzelner Makros

Eines der Hauptprobleme bei der Entwicklung von ME10-Makroanwendungen ist die Vorgehensweise des Haltens vieler Makros in einer einzelnen Datei, den bereits oben beschriebenen ME10-Moduldateien. Bei dem oben beschriebenen Projekt sind die Makros nach Modulzugehörigkeit in einzelnen Dateien organisiert, was dazu führt, dass teilweise mehrere hundert Makros in einer einzelnen Datei enthalten sind, die alle zusammengehören und in ihrer Funktionalität voneinander abhängen.

Wenn in einer Projektgruppe mehrere Entwickler gleichzeitig mit einer dieser großen Moduldateien arbeiten wollen, führt dies unweigerlich zu Problemen, da in dem Moment in dem diese Datei von einem Entwickler ausgecheckt wird, diese für alle andere Entwickler gesperrt ist und somit erhebliche Koordinationsprobleme über die gemachten Änderungen entstehen. Die Lösung für dieses Problem lautet, die große Modul-Datei zu zerlegen, so dass jedes einzelne Makro der Datei separat bearbeitet werden kann. Hierzu wird in der Datenbank des SLM-Systems eine neue ME10-Makro-Klasse von der Datei-Klasse abgeleitet, welche die einzelnen Makros der Moduldatei aufnimmt. Das Auseinandernehmen und Zusammensetzen der Makrodateien wird durch VB-Scripte vorgenommen, die in der Datenbank hinterlegt sind.

Diese Scripte führen bei einem Eincheck-Vorgang einer Moduldatei eine Analyse dieser Datei durch, um alle enthaltenen Makros zu identifizieren. Jedes ME10-Makro wird als eigenständiges Objekt der Klasse ME10-Makro in die Datenbasis des SLM-Systems aufgenommen (vgl. Abbildung 6.16).

Die einzelnen Makrodateien werden über hierarchische Links mit der Ausgangsdatei verknüpft, so dass der Zusammenhang der Dateien auch unabhängig vom Inhalt der Ausgangsdatei hergestellt werden kann. Die einzelnen Passagen in der Originaldatei, die Makros enthalten, werden durch Include-Anweisungen ersetzt, welche auf die Datei verweisen, die den herausgeschnittenen Inhalt enthält. Passagen, die nicht dem ME10-Makrocode entsprechen, wie z.B. Kommentare, bleiben davon unberührt.

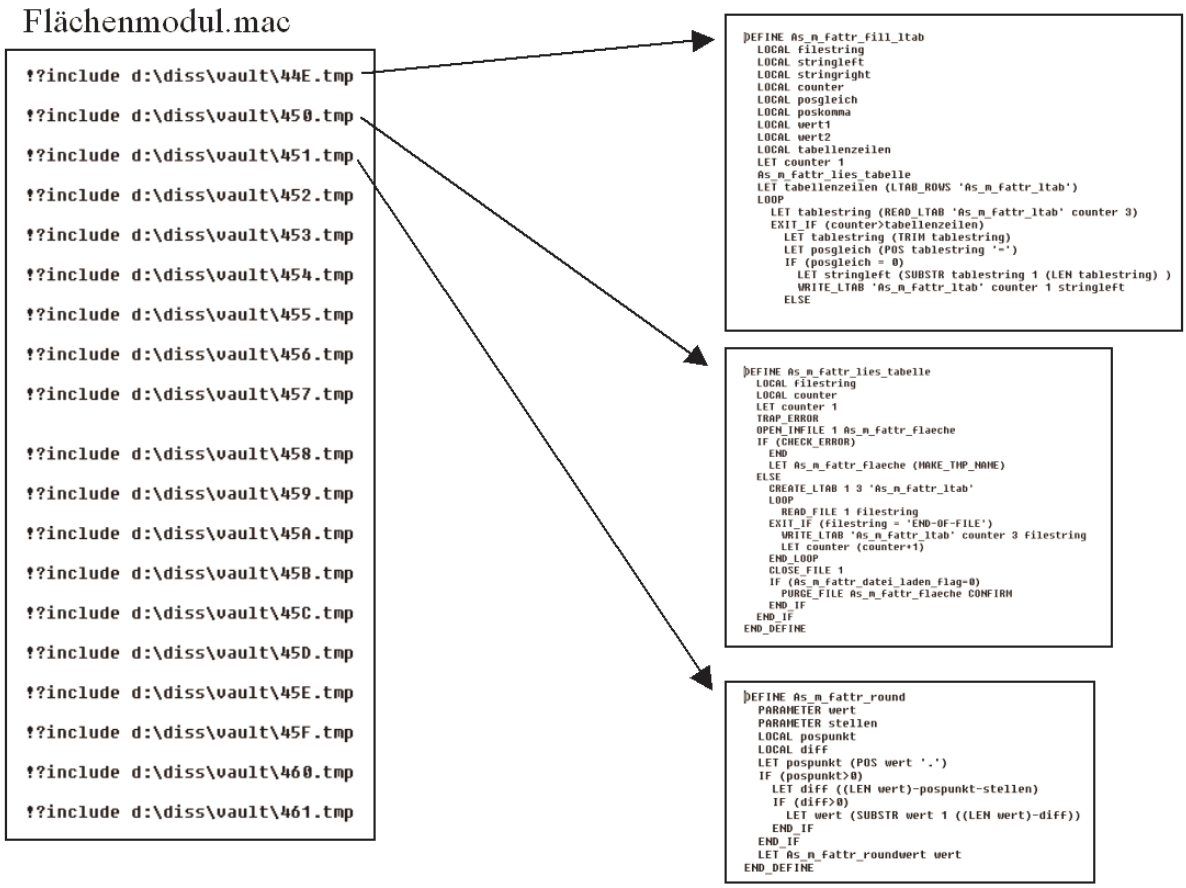


Abbildung 6.16: Zerlegung einer Moduldatei

Anschließend lassen sich die einzelnen Dateien unabhängig voneinander bearbeiten, insbesondere lässt sich jedes Makro einzeln bearbeiten. Beim neuerlichen Check-In eines einzelnen Makros wird allerdings in der Datenbank eine neue Datei angelegt und diese Datei im Vault unter einem neuen Namen abgespeichert. Deshalb muss bei einem Check-In eines einzelnen Makros die Include-Anweisung in der Ausgangsdatei durch eine Include-Anweisung ersetzt werden, die auf die neue Version des Makros verweist. Die hierarchischen Links, durch welche die Dateien verbunden sind, werden ohnehin automatisch durch das SLM-System aktualisiert.

Das Einfügen eines neuen Makros in eine Moduldatei geschieht, indem die Moduldatei als ganzes ausgecheckt, das Makro eingefügt und die Moduldatei wieder in das SLM-System eingecheckt wird. Beim erneuten Eincheckvorgang wird die Moduldatei nun durch das zuständige Skript wieder in die einzelnen Makros zerlegt. Anhand der Include-Anweisungen

in der Vorgängerversion der Modul-Datei werden die einzelnen Makros der Vorgängerversion identifiziert und mit den beim Eincheckvorgang neu erzeugten Makros verglichen. Es werden diejenigen Makros herausgefiltert, die neu sind oder verändert wurden. Entsprechend dieser Analyse werden die Include-Anweisungen in der neuen Version der Modul-Datei angepasst und nur diejenigen Makros, die neu sind oder verändert wurden, zusätzlich in die eingecheckte Modul-Datei eingetragen, ansonsten die Include-Anweisungen aus der Vorgängerversion übernommen. Die überflüssigen, beim Eincheckvorgang durch das Skript erzeugten Makros, die gegenüber der Vorgängerversion der Modul-Datei nicht neu oder verändert worden waren, werden im SLM-System verworfen und gelöscht. In einem letzten Schritt werden neue hierarchische Links erzeugt, welche die Makros mit der neuen Version Modul-Datei verknüpfen.

6.6 Vorgehensweise bei der Durchführung des Projektes PPNEU

6.6.1 Definition des Projektteams

Zunächst wird ein kleines, aber festes Team zur Realisierung des Redesign bestimmt, welches je nach Zustand der Projektbearbeitung durch andere kleine Teameinheiten ergänzt werden kann. Das SLM-System lässt eine derartige Vorgehensweise zu, ohne dass der gesamte Projektablauf dadurch gestört würde. Aus der geringen Größe des Haupt-Entwicklungsteams ergibt sich eine überschaubare Komplexität der teaminternen Koordination, so dass die folgende einfache Rollenverteilung [30] vorgenommen werden kann:

- Systemadministrator: Person die das Projekt, die Arbeitsabläufe, die Rollenverteilung im Projekt und die Metadokumente definiert
- Teamleiter (T): Erstellung von neuen Dateien, Zuweisung von Aufgaben, Freigabe von Dateien
- Entwickler (E): Mehrere Personen, welche die Quellcodedateien bearbeiten
- Tester/Prüfer (P): Testen der Funktionalität der Quellcodedateien und Prüffreigabe

Der Systemadministrator muss nicht notwendigerweise Bestandteil des Entwicklungsteams sein. Es kann sich hierbei auch um einen Spezialisten handeln, der Erfahrung mit der

Konfiguration des SLM-Systems besitzt. Auf jeden Fall aber darf die Rolle des Systemadministrator mit keiner der anderen Rollen im Team zusammenfallen, da sonst im Entwicklungsprozess unbeabsichtigte Datenverluste entstehen können. Sowohl der Teamleiter, als auch der Entwickler müssen jeweils zusätzlich die Tester-Rolle übernehmen, da die Prüfung eines Dokumentes nicht durch die Person vorgenommen werden dürfen, die vorher die Bearbeitung realisiert haben.

Das hier vorliegende Projekt beginnt normalerweise mit einer ausführlichen Konzeptstudie, die den IST-Zustand der vorhandenen Software, ein Konzept für das Redesign, sowie eine Aufwandsabschätzung beinhalten. Auf eine ausführliche Betrachtung kann hier verzichtet werden, zumal in den vorherigen Kapiteln die Inhalte einer Konzeptstudie schon wiedergegeben wurden. Der Vollständigkeit halber sollen hier nur kurz die Punkte wiedergegeben werden, die eine Konzeptstudie üblicherweise abgedeckt:

- Spezifikation der Anforderungen
- Aufnahme der bestehenden Funktionalität, sowie gewünschte Verbesserungen / Veränderungen
- Mechanismen zur Umsetzung, insbesondere zur Dokumentation der Software
- Konzept für Modulstruktur sowie Spezifikation der Module
- Zeit- und Aufwandsabschätzung für die Implementierung

Die Konzeptstudie enthält eine vollständige Funktionsliste, die als Grundlage für die programmtechnische Umsetzung dient. Vorab werden anhand der Funktionsliste die Aufgaben der Funktionsspezifikationen definiert, die Datenströme analysiert und die notwendigen Schnittstellen dargestellt. Entsprechend der Architektur des Systems wird durch den Teamleiter und den Entwickler eine Funktionsspezifikation der Benutzeroberfläche erstellt. Wenn diese weit genügend fortgeschritten ist, beginnt der Entwickler mit der Implementierung, während der Teamleiter die Spezifikation vervollständigt und anschließend mit der Funktionsspezifikation der logischen Ebene beginnt.

6.6.2 Freigabeschema

Die folgenden Stati (Tabelle 6.2) werden gemäß den Vorgaben des SLM-Systems für das PPNEU-Projekt vorgesehen und können nur von den jeweils hier angegebenen Teammitgliedern durchgeführt werden:

Status	**Beschreibung**	**Personen (vgl. 6.6.1)**
New	Anlage eines neuen Dokumentes im SLM-System	(T)
Checked In	Übergabe eines Dokumentes an den Vault	(T,E)
Checked Out	Herunterladen und Sperren eines Dokumentes	(T,E)
Test release	Dokument ist zur Prüfung freigegeben	(T)
Tested	Prüfen eines Dokumentes auf Richtigkeit	(T)
Released	Freigabe eines Dokumentes zu Verwendung	(T)
Obsolete	Die Datei ist veraltet	(SLM-System)
Disabled	Dokument darf nicht verwendet werden	(T)

Tabelle 6.2: Stati im Projekt PPNEU

Den Status „Obsolete“ bekommt ein Objekt im SLM-System automatisch zugewiesen, sobald eine neuere Version des Objektes vorliegt. Dies geschieht, da prinzipiell jedes Objekt in verschiedenen Projekten referenziert sein kann, innerhalb dieses anderen Projektes aber eine ältere Version des Objektes Verwendung finden muss, da das Projekt beispielsweise mit der neuen Version nicht mehr kompatibel ist. Bei der Bearbeitung dieses Projektes sieht der Bearbeiter nun, dass die verwendete Version veraltet ist und möglicherweise durch eine neuere Version ersetzt werden kann, was aber im Einzelfall zu Prüfen ist.

Aus diesen Vorgaben lässt sich nun das folgende verfeinerte Freigabeschema (vgl. Abbildung 6.17 und Abbildung 5.12) erstellen. Es wurde das in Kapitel 5.3.3 beschriebene einfache Freigabeschema um eine Testphase erweitert, welche die Vorlage eines Dokumentes einer Testperson im Team vorschreibt, bevor die endgültige Freigabe erfolgen kann.

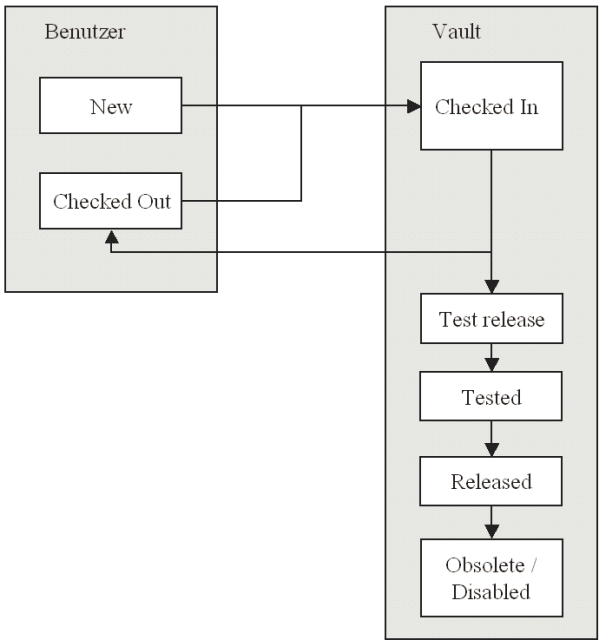


Abbildung 6.17: Angepasstes Freigabeschema für das Projekt PPNEU

6.6.3 Prozessdefinition für das Projekt PPNEU

Für die Vereinfachung der Kooperation im Team und den damit verbundenen Koordinationsmöglichen werden die folgenden Prozesse (vgl. Abbildung 6.18) spezifiziert:

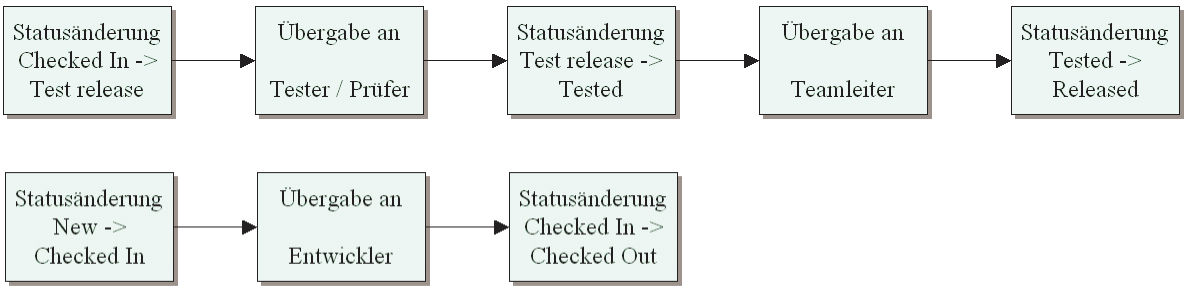


Abbildung 6.18: Freigabeprozess (oben) und Neu-Anlage-Prozess (unten)

Die Definition der Prozesse erfolgt analog Kapitel 5.3.5. Der obere Prozess ist eine Abbildung des zweistufigen Freigabeprozesses (vgl. Abbildung 6.17) und sorgt für die automatische Weiterleitung der entsprechenden Dateien innerhalb des Entwicklungsteams, die ansonsten

manuell oder über persönliche Absprache erfolgen müsste. Er wird verwendet, sobald ein Entwickler mit der Arbeit an einer Datei oder mehrerer zusammenhängender Dateien (z.B. C-Quellcodedatei und der zugehörigen Headerdatei) fertig ist. Der Entwickler erzeugt einen neuen Prozess aus der Prozessvorlage, verknüpft die notwendigen Dateien mit dem Prozess und startet diesen. Gemäß der Prozessdefinition werden die hierarchisch mit dem Prozess verknüpften Dateien zunächst mit dem Status „Test release“ versehen und dann der Prozess and die Gruppe der Tester des Projektes weitergeleitet. Hier übernimmt nun ein Mitglied aus dieser Gruppe den Prozess und begutachtet die Dateien. Erfüllen sie die Testkriterien, so kann der Tester den Prozess fortsetzen, wobei allen mit dem Prozess hierarchisch verknüpften Dateien der Status „Tested“ zugewiesen und der Prozess an den Teamleiter weitergeleitet würde, oder ihn abweisen, wobei der Entwickler den Prozess zurückerhalten würde und die Dateien erneut zu bearbeiten hätte. Hat der Teamleiter den Prozess übernommen, so kann er die Dateien entweder selbst noch einmal prüfen und gegebenenfalls den Prozess mit einer Anmerkung zurückweisen oder den Prozess zu Ende laufen lassen, so dass sie Dateien freigegeben werden. Der untere Prozess (vgl. Abbildung 6.18) funktioniert analog zu dem oben beschriebenen.

Das Projekt für das Redesign von ProfiPlus gliedert sich nun in mehrere Entwicklungsphasen, die sich an der Architektur der Software orientieren. Der Aufbau von PPNEU erfolgt in mehreren Schichten (vgl. Abbildung 6.5), die unabhängig voneinander sind und deshalb nacheinander implementiert werden können.

6.6.4 Implementierung der Benutzerebene

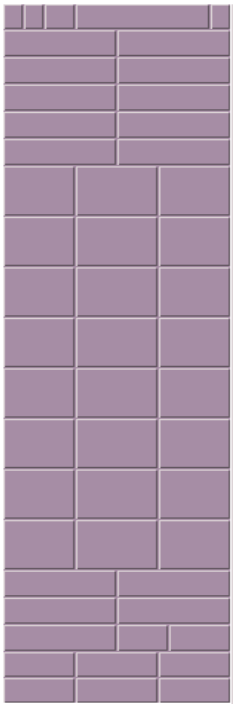
Die Implementierung der Benutzerebene ist der einfachste Teil der Implementierung des Gesamtsystems. Durch die strikte Trennung der einzelnen Ebenen der Architektur kann sie unabhängig von den anderen Teilen des Systems erfolgen. Für die Implementierung von PPNEU können im wesentlichen die bereits vorhandenen Menüdefinitionen des alten Systems PP2000 als Vorlage verwendet werden, lediglich die Abstraktion von der Funktionsebene muss durchgeführt werden.

Die Implementierung der Benutzerebene beinhaltet die in Tabelle 6.3 dargestellten Schritte, inklusive der zu erstellenden Dateien (vgl. Abbildung 6.6):

Schritt	Beschreibung	Datei
1.	Erstellung des Layouts aller Menus	pp_menu_pe.mac
2.	Erstellung einer Liste aller Meldungen und Texte im PPNEU und Anlage einer Sprachdatei. Die Erstellung erfolgt in enger Anlehnung an die Meldungen im PP2000.	pp_lang.mac
3.	Erstellung der in den Menus darzustellenden Symbole	pp_icons.mac
4.	Erstellung des Logikmoduls der Oberfläche	pp_ui.mac
5.	Erstellung der eigentlichen Benutzeroberfläche, in der in den erzeugten Menulayouts die Texte und Symbole der PPNEU-Oberfläche angezeigt werden und mit den entsprechenden Aufrufen aus dem Logikmodul hinterlegt werden.	pp_menus.mac

Tabelle 6.3: Übersicht der Schritte zu Erstellung der Benutzeroberfläche

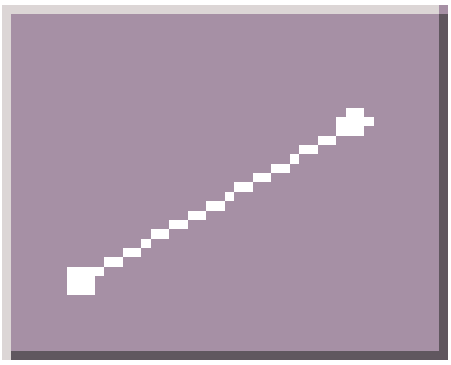
Zunächst wird das Layout für alle Bildschirmmenus entworfen (vgl. Abbildung 6.19). Die Höhe der einzelnen Felder wird in Punkten der Bildschirmauflösung angegeben und ist in der Variable „text_slot_height“ gespeichert, die bei der Initialisierung des Systems errechnet wird. Dies ist variabel gestaltet, da mit unterschiedlicher Bildschirmauflösung die Höhe der Menus angepasst werden muss. Die Aufteilung der Felder des Menus geschieht über Zeichenketten aus Leerzeichen in denen die einzelnen Felder durch das Zeichen ‚|’ getrennt werden. Die Breite des Menus hängt von der Größe der eingestellten Schriftart im ME10 ab und ergibt sich somit aus der Breite der Leerzeichen. Nach diesem Schritt enthalten die Menus weder Texte oder Symbole, noch sind Funktionsaufrufe hinterlegt.



```
DEFINE ppneu_m_cre_create_layout
PARAMETER  ppneu_p_menuename
CURRENT_MENU ppneu_p_menuename
MENU_LAYOUT  Menu_position
 (Headline_height)     ' | |  |             | '
 (text_slot_height+1)  '          |          '
 (text_slot_height+1)  '          |          '
 (text_slot_height+1)  '          |          '
 (text_slot_height+1)  '          |          '
 (text_slot_height+1)  '          |          '
 (text_slot_height*2)  '       |      |      '
 (text_slot_height*2)  '       |      |      '
 (text_slot_height*2)  '       |      |      '
 (text_slot_height*2)  '       |      |      '
 (text_slot_height*2)  '       |      |      '
 (text_slot_height*2)  '       |      |      '
 (text_slot_height*2)  '       |      |      '
 (text_slot_height*2)  '       |      |      '
 (text_slot_height+1)  '          |          '
 (text_slot_height+1)  '          |          '
 (text_slot_height+1)  '          |    |     '
 (bottom_slot_height)  '       |      |      '
 (bottom_slot_height)  '       |      |      '
  Menu_transfer_vector
END_DEFINE
```

Abbildung 6.19: Menu Layout „Erstellen“ und zugehöriger Makro-Quellcode

Zur Hinterlegung der Felder des Menus, müssen nun die Symbole entworfen werden, was über zusätzliche Werkzeuge aus ME10-Zeichnungen geschieht. Die Erstellung der Symbole erfolgt über Hilfsprogramme, die den dargestellten Code aus ME10-Zeichnungen extrahieren. Die Codierung erfolgt folgendermaßen: #255#x1#y1#x2#y2#x3#y3... . Eine jede solche Sequenz beginnt immer mit „#255“ und wird gefolgt von einer Folge aus X- und Y-Koordinaten. Sie erzeugt jeweils einen Liniezug von (x1,y1) nach (x2,y2) nach (x3,y3) bis nach (xn,yn) innerhalb eines Feldes.



```
DEFINE Ppneu_vi_2pkte
(#255#13#9#22#9#22#12#255#22#29\
#13#29#13#9#255#22#25#31#25#31\
#12#22#12#22#29#255#22#25#22#12\
#255#31#9#35#9#35#29#31#29#31\
#9#255#11#19#13#19#255#14#19#15\
#19#255#16#19#19#19#255#20#19#20\
#19#255#21#19#24#19#255#25#19#25\
#19#255#26#19#29#19#255#30#19#30\
#19#255#31#19#34#19#255#35#19#35\
#19#255#36#19#37#19)
END_DEFINE
```

Abbildung 6.20: Symbol für "Linie zwei Punkte" und zugehöriger Code

Die darzustellenden Texte werden in Variablen gespeichert, damit auf einfache Weise multilinguale Versionen der PPNEU-Software erstellt werden können (vgl. Abbildung 6.21).

```
     LET m1 ' ERSTELLEN'
     LET m2 'LINIEN    '
     LET m3 'MITTELLIN '
     LET m4 'KREISE    '
     LET m5 'SPLINES   '
     LET m6 'ATT SETZEN'
     LET m7 'ATT ÜBERTR'
     LET m8 'VERSAT'
     ...
     ...
```

Abbildung 6.21: Darzustellende Texte der PPNEU-Menus (Variablen „m1“ bis „mx“)

Die Erstellung der Logik-Ebene der Benutzeroberfläche beinhaltet die Implementierung der Mechanismen für das Reset-Makro (vgl. Kap. 0), sowie die Aufrufe der eigentlichen ME10-Funktionalität und der noch zu erstellenden PPNEU-Funktionalität. Nun sind die Menus mit den Symbolen und Texten zu hinterlegen, sowie mit den Aufrufen der Makros aus der Logik-Ebene der Benutzeroberfläche.

```
DEFINE Ppneu_m_create_1
  LOCAL Ppneu_v_zeile
  IF (I_port)
    Check_i_port
  END_IF

  IF (NOT I_port)
    LET Lastmen 'Ppneu_m_create_1'
    CURRENT_MENU Ppneu_ml_create_1_name
    LET Lastlay Ppneu_ml_create_1_name T_clear_menu Ppneu_m_menu_control_icons
    'Sm_create_1' Ppneu_m_create_allg
    MENU Colo0 Bcol5 m1 '' 1 4
    LET Ppneu_v_zeile 6                 {anfangen in Zeile 6}
...
    MENU Colo0 Bcol5 m1 '' 1 4
...
    LET Ppneu_v_zeile (Ppneu_v_zeile+1)  {nächste Zeile}

    MENU Colo1 Bcol0 Ppneu_vi_2pkte 'Ppneu_logik_line_two_pts' Ppneu_v_zeile 1
    MENU Colo1 Bcol0 Ppneu_vi_horiz 'Ppneu_logik_line_horiz' Ppneu_v_zeile 2
...
...
END_DEFINE
```

Abbildung 6.22: Menubelegung für das "Erstellen"-Menu von PPNEU

In der markierten Code-Zeile aus der Belegung für das „Erstellen“-Menu (vgl. Abbildung 6.22) wird dem Feld in der Zeile Ppneu_v_zeile und der Spalte 1 des Menu-Layouts

(vgl. Abbildung 6.19) das Symbol „Ppneu_vi_2pkte“ (vgl. Abbildung 6.21) zugewiesen, in der Vordergrundfarbe Colo1 und der Hintergrundfarbe Bcol0, sowie der Funktionsaufruf „Ppneu_logik_line_two_pts“ hinterlegt, der ausgeführt wird, sobald der Benutzer auf dieses Feld „klickt“. Nach Belegung aller Felder ergibt sich das vollständige Menu (vgl. Abbildung 6.1).

Dem Auftraggeber kann nach Fertigstellung aller Menus ein Design-Prototyp zur Verfügung gestellt werden, an dem dieser überprüfen kann, ob die Benutzeroberfläche seinen Anforderungen entspricht. Für das ordnungsgemäße Funktionieren dieses Prototyps, muss eine Zusammenstellung aller Makros der PPNEU-Funktionalität, die aus dem Logikmodul der Oberfläche heraus aufgerufen werden, in einer gesonderten Makrodatei erfolgen, da es beim Aufruf nicht vorhandener Makros zu Fehlermeldungen kommt. Diese Makros sind leer und enthalten keinen Code. Die reinen ME10-Funktionalitäten, die nicht Bestandteil von PPNEU sind, können mit dem Prototypen bereits getestet werden, da sie im Logikmodul implementiert sind.

6.6.5 Implementierung der Funktionsebene

Die Implementierung der Funktionsebene bildet den eigentlichen, über den Funktionsumfang von ME10 hinaus gehenden Teil von PPNEU ab. Zentraler Bestandteil der Funktionsebene ist die so genannte Toolbox, die vor der Implementierung der eigentlichen Benutzerfunktionalität der Funktionsebene zu erstellen ist.

Das Modul Toolbox ist ein PPNEU internes Modul, welches Dienstleistungsfunktionen für andere Module (vgl. Abbildung 6.5) von PPNEU bereitstellt. Durch die zentrale Bereitstellung von oft benötigten Basisfunktionen wird die Wiederverwendung von bereits implementierten Algorithmen gesteigert. Diese Möglichkeit der Wiederverwendung reduziert gleichzeitig den Erstellungsaufwand für die Funktionsmodule erheblich. Die Tabelle 6.4 gibt eine Übersicht über die zu erstellenden Dateien der Toolbox:

Datei	Beschreibung
pp_konvert.mac	Funktionen zur Umwandlung von Skalaren in Vektoren, von Punkten in Einheiten usw.
pp_tool.mac	Alle internen Funktionalitäten der Toolbox, wie der Zugriff auf Konfigurationsdateien oder Tabellen, die keine Rückgabewerte liefern.
pp_tool.dll	Alle internen Funktionalitäten der Toolbox die Rückgabewerte liefern. Sie müssen deshalb über die Programmierschnittstelle von ME10 als Bibliothek realisiert werden.
pp_io.mac	Alle Funktionalitäten der Toolbox zur Interaktion mit dem Benutzer.

Tabelle 6.4: Dateien der Toolbox

Nach der Erstellung der Hauptfunktionalitäten der Toolbox erfolgt die Erstellung der eigentlichen Funktionalität von PPNEU in den einzelnen Modulen, die der Reihe nach abgearbeitet werden. Im Rahmen der Erstellung der Module erfolgt ebenfalls die Erstellung der zugehörigen Konfigurationsdateien, inklusive der Anlage der notwendigen Wertzuweisungen darin.

Die Erstellung der einzelnen Module durch die Entwickler kann durch die Trennung von Oberfläche und Funktionalität unabhängig vom Gesamtzusammenhang erstellt und getestet werden. Lediglich die Toolbox ist für die Erstellung der einzelnen Module der Funktionsebene von PPNEU entwicklerseitig notwendig. Im Rahmen dieses Entwicklungsprozesses, werden alle weiteren allgemeingültigen Funktionalitäten, die als sinnvoll und universell identifiziert wurden, ebenfalls in die Toolbox integriert. Nach Abschluss der Entwicklung der Funktionsebene kann jedes Modul daraus einzeln auf seine korrekte Funktionalität entsprechend des Pflichtenheftes und der Dokumentation des alten PP2000 getestet werden.

6.6.6 Integration des Systems

Die Integration des Systems erfordert das Zusammenspiel aller erstellten Module und der Oberfläche als Ganzes. Hierzu muss entsprechend Kapitel 6.4.1, die Implementierung des

Start- und Initialisierungsvorganges erfolgen. Für den Startprozess muss ein Makro-Script erstellt werden, welches eingebunden in den Startvorgang von ME10, das PPNEU automatisch lädt und initialisiert (vgl. Abbildung 6.9). Dieser Lade- und Initialisierungsvorgang muss nach Abschluss der Integration fehlerfrei funktionieren. Als nächstes muss getestet werden, ob von der Oberfläche des Systems alle Funktionalitäten von ME10 und PPNEU ordnungsgemäß und fehlerfrei aufgerufen werden können. Dieser Test ist in einem Protokoll festzuhalten. Fehler, die hierbei auftreten können, sind:

- Nicht alle Funktionalitäten der Oberfläche sind in den Modulen implementiert
- Unterschiedliche Schreibweisen von Funktionen
- Die Aufrufreihenfolge ist nicht korrekt oder unvollständig
- Die übergebenen Parameter sind unvollständig oder haben abweichende Datentypen
- Die Parameter in den Konfigurationsdateien sind unvollständig oder inkorrekt
- Die Symbole werden nicht in der richtigen Größe dargestellt

Vor Beendigung der Integration müssen alle diese Probleme behoben werden. Nach der Durchführung der Integration steht nun ein vollständiger Funktionsprototyp zur Verfügung, der nun anschließend einer Test- und Fehlerbehebung unterzogen werden kann.

6.6.7 Test und Fehlerbehebung

Der Funktionstest läuft in mehreren Schritten ab, wobei die Durchführung des Testes durch das gesamte Entwicklungsteam geschieht, inklusive aller Entwickler, Tester und des Teamleiters, damit bei auftretenden Problemen die Entwickler wissen, wie getestet wird und die Tester wissen, wie die Software „gedacht“ ist:

- Erstellung geeigneter Testszenarien, in Absprache mit dem Kunden, anhand des Handbuches von PP2000, vorgegebener ME10-Zeichnungen, vorgegebener Software- und Betriebssystem-Versionen und bestimmter bei Industriekunden häufig vorkommender Konfigurationen.

- Überprüfung, ob alle Anforderungen und Anwendungsfälle durch die Testszenarien abgedeckt sind.
- Der Test wird mit den ME10-Versionen 8.7 bis 11.5 durchgeführt, jeweils im „PE-Look“ und im „Mixed Look“. Der „Mixed-Look“ erlaubt es, Menus im PE-Look unter der Windowsoberfläche von ME10 darzustellen.
- Der Test muss jeweils unter Windows und UNIX durchgeführt werden, wobei insbesondere darauf zu achten ist, dass die Zahlenbehandlung unter Windows und UNIX teilweise unterschiedlich ist, da andere Rundungsroutinen verwendet werden und beispielsweise Divisionsergebnisse unter Windows korrekt sein können, während sie unter UNIX in der letzten Nachkommastelle fehlerhaft sind.
- Es ist bei verschiedenen Bildschirmauflösungen zu testen, ob alle Benutzer-interaktionen in der jeweiligen Bildschirmauflösung korrekt und fehlerfrei dargestellt werden. (800x600, 1024x768, 1280x1024, usw.)

Sämtliche durchgeführten Tests werden in einem vorgegebenen Testprotokoll festgehalten, welches im SLM-System gehalten wird. Aus sich ergebenden Fehlern werden Änderungs-listen im SLM-System erzeugt und mit den zu ändernden Dateien verlinkt, damit die Entwickler über alle notwendigen Änderungen informiert sind. Zu jeder Änderung ist eine genaue Fehlerbeschreibung beizufügen, mit einer Anleitung zur Reproduktion des Fehlers.

In den Testszenarien ist jede Funktionalität mindestens einmal auf Korrektheit zu testen, anhand der vorgesehenen Bedienungsweise [31]. Weiterhin ist bei jeder Datenabfrage zu testen, ob das System bei der Eingabe unsinniger, falscher oder inkorrekter Daten stabil läuft. Wie reagiert das System z.B. bei Abfrage einer positiven Ganzzahl auf die Eingaben eines negativen Wertes, einer Kommazahl oder eines Textes.

Weiterhin ist zu prüfen: Wie reagiert das System bei Permutationen der korrekten Aktionsfolge durch den Benutzer (Der Benutzer macht die Eingaben in anderer Reihenfolge, als vom System vorgesehen)? Ist die Benutzerführung gut?

Diese gesamte Testprozedur ist iterativ zu wiederholen, bis die Software im Rahmen der Testszenarien fehlerfrei funktioniert, wobei sich jeder Testphase jeweils eine Fehlerbehebung

anhand der erstellten Fehlerprotokolle und Fehlerbeschreibungen anschließt. Nach befriedigendem Abschluss der Testphase, kann das Projekt als Ganzes freigegeben werden und die Auslieferung an den Kunden erfolgen.

6.6.8 Auslieferung an den Kunden

Das Redesign von ProfiPlus führte zu einer besser wartbaren, strukturierten, fehlerfreien und gut dokumentierten Version von ProfiPlus, die nun der vertreibende Lieferant seinen anwendenden Kunden zur Verfügung stellen kann. Die Auslieferung an den Lieferanten umfasst im wesentlichen vier Schritte [32]:

1. Abnahme der PPNEU-Software
2. Abnahme der Quellen und der Entwicklerdokumentation
3. Installation des SLM-Systems und der Entwicklungsumgebung beim Kunden und Auslieferung der gesamten Datenbank, sowie des Vaults des PPNEU-Projekts.
4. Einweisung des Kunden in das System

Die Abnahme der PPNEU-Software erfolgt entsprechend des Pflichtenheftes und der vereinbarten Testszenarien. Die Tests werden durch den Kunden im Beisein des Teamleiters durchgeführt und mitprotokolliert. Die Tests sind fehlerfrei abgeschlossen und das Abnahmeprotokoll ist beiderseitig abgezeichnet. Nach der Abnahme der PPNEU-Software erfolgt nun die Abnahme der Quellen und der Dokumentation des Systems. Hierbei ist zu prüfen, ob die Dokumentation im Sinne des Kunden vollständig ist, alle Quellen vorliegen und die Architektur der Software den Vereinbarungen entspricht. Diese Prüfung wird ebenfalls in einem Abnahmeprotokoll festgehalten.

Nach der eigentlichen Abnahme der PPNEU-Software durch den Kunden erfolgt in einem weiteren Schritt die Installation des SLM-Systems beim Kunden, die Einrichtung der Entwickler-Arbeitsplätze, die Installation eines Datenbankservers und Übernahme der Entwicklungsdatenbank, sowie schließlich die Einrichtung eines Vault-Servers und die Übernahme aller Dateien der Entwicklungsversion in den Vault. Die Funktionsfähigkeit des Systems wird in einem weiteren Abnahmeprotokoll festgehalten.

Nach der Abnahme erfolgt durch den Teamleiter und gegebenenfalls durch beteiligte Entwickler die Einweisung der mit der Wartung der PPNEU-Software beauftragten Mitarbeiter des Kunden in das SLM-System und die Entwicklungsumgebung. Die Einweisung wird mit einer Abnahme durch den Kunden abgeschlossen. Nach der Einweisung obliegt die weitere Wartung der PPNEU-Software dem Kunden und das PPNEU-Projekt ist abgeschlossen.

7 Zusammenfassung und Ausblick

In dieser Arbeit wird ein Konzept entwickelt und beispielhaft implementiert, welches den Lebenszyklus von Softwareprodukten abbildet. Dieses wird Software-Lifecycle-Management-System genannt (SLM-System).

Die Untersuchung der Softwareentwicklung in mehreren Firmen aus verschiedenen Sparten hat gezeigt, dass sich die bestehenden Werkzeuge zur Unterstützung des Entwicklungsprozesses häufig nur auf die eigentliche Planung und Erstellung der Software beschränken. Dabei wird häufig in kleinen und mittelständischen Unternehmen der Bereich der Dokumentation und Wartung der Software vernachlässigt, ebenso wie deren Betrachtung als Produkt. Hinzu kommt, dass insbesondere in einigen Nischen der Softwareerstellung, wie z.B. der Entwicklung von Software im CAD-Bereich, nicht einmal brauchbare Werkzeuge existieren, die den Entwicklungsprozess in geeigneter Weise unterstützen.

Es wird ein flexibel einsetzbares Werkzeug zur Verfügung gestellt, welches ein beliebiges, gewähltes Vorgehen bei der Softwareentwicklung unterstützt. Hierbei liegt ein besonderes Augenmerk auf dem gesamten Entwicklungsprozess, vom Auftragseingang bis hin zur Wartung der Software, bei der generell die meisten Kosten entstehen. Weiterhin wird die Softwareentwicklung in bisher nicht unterstützen Bereichen verbessert, durch die Möglichkeit der Einbindung geeigneter Entwicklungswerkzeuge.

Die Unterstützung einer parallelen Softwareentwicklung in einem Entwicklungsteam ist durch das integrierte Workflow-Management und das flexible Lebenszyklusmodell gegeben. Durch die Verwendung der .NET-Architektur wird das SLM-System plattformunabhängig und der Einsatz von Webservices gestattet die Verwendung des SLM-Systems in heterogenen Netzwerken, ohne durch so genannte Firewalls behindert zu werden, die gegebenenfalls direkte Funktionsaufrufe über das Internet blockieren.

Anhand des Einsatzes des SLM-Systems beim Redesign der Software „ProfiPlus“ wurde das Konzept des SLM-Systems in der Praxis erprobt und hat seine Tauglichkeit zur Unterstützung

des Entwicklungsprozesses gezeigt. Insbesondere wurde daran die Eignung des SLM-Systems zur Unterstützung des Softwareentwicklungsprozesses in Bereichen ersichtlich, in denen es bis jetzt kaum oder gar keine Unterstützung gab, und die Möglichkeit, als flexible Basis zur Integration von beliebigen Softwareentwicklungsumgebungen zu dienen.

Für die weitere Entwicklung SLM-Systems ist es notwendig, vorkonfektionierte Schnittstellen zu Standard-CASE-Tools zu entwickeln, wie etwa MS Visual Studio oder Rational Rose, aber auch zu Office-Anwendungen, die eine einfache und kostengünstige Integration des SLM-Systems in bestehende Entwicklungs-Landschaften ermöglichen. Ebenso ist es wünschenswert, gängige Vorgehensmodelle bei der Softwareentwicklung durch vorgegebene Freigabeschemata und Workflowmodelle in Projektvorlagen zu integrieren, sowie vordefinierte Metadokumente standardmäßig im SLM-System vorzuhalten, um den anfänglichen Konfigurationsaufwand beim Einsatz des Systems zu minimieren.

8 Anhang

8.1 Literaturverzeichnis

[1] N.N.: *Methodik zum Entwickeln technischer Systeme und Produkte*, VDI-Richtlinie 2221, VDI-Verlag, Düsseldorf 1993

[2] Rainer Dumke: *Software Engineering*, Vieweg Verlag, ISBN 3-528-25355-X, 2001

[3] Spur, G.; Krause, F.: *Das virtuelle Produkt*, Hanser Verlag München, ISBN 3-446-19176-3,1997

[4] Helmut Balzert: *Lehrbuch der Softwaretechnik/ Band 2/ Software-Management/ Software-Qualitätsmanagement/ Unternehmensmodellierung*, Spektrum Verlag, ISBN 3-8274-0042-2, 1998

[5] Werner u.a.: *Taschenbuch der Informatik*, Fachbuchverlag Leipzig, ISBN 3-343-00892-3, 2. Auflage 1995

[6] R. Buck-Emden: *Die Technologie des SAP R/3 Systems*, Addison-Wesley, ISBN 3-8273-1379-1, 4. Auflage 1999

[7] N.N.: COBOL 85 V2.1, Siemens Nixdorf, Auflage August 1993

[8] Matzke, Bernd: ABAP/4; Die Programmiersprache des SAP-Systems R/3, Addison-Wesley, ISBN 3-8273-1372-4, 2.Auflage 1998

[9] Helmut Balzert: *Lehrbuch der Softwaretechnik/ Band 1/ Software-Entwicklung*, Spektrum Verlag, ISBN 3-8274-0042-2,1996

[10] Borchart, Christian: *Entwicklung und Anwendung eines Vorgehensmodells zur Unterstützung des objektorientierten Softwareentwicklungsprozesses am Beispiel eines CAD-Systems*, Shaker-Verlag, ISBN 3-8265-3331-3, 1998

[11] Helmut Weber: Praktische Systemprogrammierung; Grundlagen und Realisierung unter UNIX und verwandten Systemen, Vieweg Verlag, ISBN 3-528-05658-4, Auflage 1997

[12] Frank Lobeck: *Konzept für ein objektorientiertes, bereichsübergreifendes Dokumenten-informations- und verwaltungs- system*, Shaker Verlag, ISBN 3-8265-6639-4, 1. Auflage 1999

[13] Elmasri; Navathe: *Fundamentals of Database Systems*, Benjamin / Cummins Publishing Company, ISBN 0-8053-0145-3, 1. Auflage 1989

[14] Hohenstein; Lauffer; Schmatz; Weikert: *Objektorientierte Datenbanksysteme*, Vieweg Verlag, ISBN 3-528-05501-4, 1996

[15] A. Kemper; A. Eickler: *Datenbanksysteme / Eine Einführung*, Oldenbourg Verlag, ISBN 3-486-24136-2, 2. Auflage 1997

[16] J. Richter; F. Balena: *Microsoft .NET Framework Programmierung mit Visual Basic .NET*, Mirosoft Press, ISBN 3-86063-682-0, 1. Auflage 2003

[17] Andreas & Peter Solymosi: *Effizient Programmieren in C# und .NET*, Vieweg, ISBN 3-528-05778-5, 1. Auflage 2001

[18] A. Moos; G. Daues: *Datenbank-Engineering*, Vieweg Verlag, ISBN 3-528-1583-8, 2. Auflage 1997

[19] Günter Bengel: Verteilte Systeme; Client-Server-Computing für Studenten und Praktiker, Vieweg Verlag, ISBN 5-328-15738-0, 2. Auflage 2002

[20] R. Dippold, A. Meier, A. Ringgenberg, W. Schnider, K. Schwinn: *Unternehmensweites Datenmanagement*, Vieweg Verlag, ISBN 3-528-25661-3, 3. Auflage 2001

[21] Platt, David S.: *Microsoft .NET / Eine Einführung*, Microsoft Press, ISBN 3-86063-690-1, 2. Auflage

[22] Kleinschmidt, Peter: *Relationale Datenbanksysteme,* Springer Verlag, ISBN 3-540-42413-X, 2. Auflage 2002

[23] Schütten, Markus: *Konzept eines COM-basierten Technischen Produktinformations-systems*, http://miless.uni-essen.de/servlets/DerivateServlet/Derivate-10481/tpis.pdf, 2001

[24] N.N.: *Visual Basic .NET Online Hilfe*, Microsoft, 2002

[25] Ronald Schnetzer: *Workflow-Management*, Vieweg Gabler, ISBN 3-528-05718-1, 1999

[26] Michael Kofler: *Visual Basic .NET / Grundlagen*, Addison-Wesley, ISBN 3-8273-1982-X, 2002

[27] Leimhofer, Gerald: *Computer Aided Software Engineering*, www.cosy.sbg.ac.at/~pmm/teaching/ SS02/se/papers/0020508.pdf, 2002

[28] Beneke, Frank: *Konzeptionelle Ansätze einer prozessorientierten Produktentwicklung*, Shaker Verlag, ISBN 3-8322-1320-5, 1. Auflage 2003-06-25

[29] Frank Rieg; Reinhard Hackenschmidt: *Softwaretechnik für Ingenieure*, Hanser Verlag, ISBN 3-446-21646-4, 2001

[30] Anderegg, Brigitte: *IT-Prozessmanagement*, Vieweg-Verlag, ISBN 3-528-5744-0, 2000

[31] Johannes Siederseben (Hrsg.): *Softwaretechnik / Praxiswissen für Softwareingenieure*, Hanser Verlag,ISBN 3-446-21843-2, 2. Auflage 1999

[32] Carl Steinweg: *Projektkompass Softwareentwicklung*, Vieweg Verlag, ISBN 3-528-35490-9, 4. Auflage 2002

8.2 Abbildungsverzeichnis

Abbildung 1.1: Produktentwicklungsprozess [3] 12
Abbildung 2.1 : Idealisierter Softwarelebenszyklus [2] 17
Abbildung 3.1: Phasenverlauf beim Softwarelebenszyklus [2] 40
Abbildung 3.2: Prozess-Definition 49
Abbildung 4.1: Abstraktion des Datenzugriffs durch ADO [21] 74
Abbildung 5.1: Aufbau des SLM-Systems 76
Abbildung 5.2: Netzwerkstruktur des SLM-Systems 77
Abbildung 5.3: allgemeine (Verzeichnis-)Struktur eines Projektes 81
Abbildung 5.4: verschiedene Arten von Links 82
Abbildung 5.5: Objektstruktur und Verlinkung von Objekten 85
Abbildung 5.6: Versionisierung von Objekten mit zugehöriger Tabellenstruktur 86
Abbildung 5.7: Darstellung der Datenbankstruktur 88
Abbildung 5.8: Start des SLM-Server-Dienstes (schematisch) 90
Abbildung 5.9: Hauptkomponenten des SLM-Systems (Datenzugriff und Datenaustausch) 92
Abbildung 5.10: ADO .NET Architektur [24] 93
Abbildung 5.11: Klassenhierarchie für Tabellenzugriff (UML) 95
Abbildung 5.12: einfaches Freigabeschema 97
Abbildung 5.13: Zustandsänderung 99
Abbildung 5.14: Tabellenstruktur zur Abbildung der Funktionalität 101
Abbildung 5.15: Prozessdefinition (Schematisch) vor Prozessstart 103
Abbildung 5.16: Prozess (schematisch) nach Durchlauf 105
Abbildung 5.17: Konfiguration einer Installationsversion (schematisch) 107
Abbildung 5.18: Zuordnungstabelle für Installationslink (beispielhaft) 108
Abbildung 5.19: Anmeldung von Clients 110
Abbildung 5.20: Klassen des SLM-Servers 114
Abbildung 5.21: Projektdialog 116
Abbildung 6.1: Standardoberfläche von ME10 und ProfiPlus-Erweiterung. 130
Abbildung 6.2: Verzeichnisstruktur von PP2000 (ausschnittsweise) 132
Abbildung 6.3: Entwicklung an PP2000 (bisherige Vorgehensweise). 135
Abbildung 6.4: Vorgehensweise bei der Verschlüsselung von Kundenmakros 136

Abbildung 6.5: Schematische Modulstruktur von PPNeu 143
Abbildung 6.6: Übersicht der Module und Dateien 144
Abbildung 6.7: Dokumentationsdatei (Neuanlage Modul-Datei) 148
Abbildung 6.8: Dokumentationsdatei (Neuanlage ME10-Makrodatei) 148
Abbildung 6.9: Startvorgang für PPNEU 149
Abbildung 6.10: Ausführung von Makros 150
Abbildung 6.11: Ini-Dateien von PPNEU (Admin.ini) 152
Abbildung 6.12: Trace Funktion für AI-Module (C-Quelltext) 155
Abbildung 6.13: Auszug der Datei pp_lang.mac 156
Abbildung 6.14: Ultra-Edit 158
Abbildung 6.15: Werkzeug-Konfiguration in Ultra-Edit 159
Abbildung 6.16: Zerlegung einer Moduldatei 163
Abbildung 6.17: Angepasstes Freigabeschema für das Projekt PPNEU 167
Abbildung 6.18: Freigabeprozess (oben) und Neu-Anlage-Prozess (unten) 167
Abbildung 6.19: Menu Layout „Erstellen“ und zugehöriger Makro-Quellcode 170
Abbildung 6.20: Symbol für "Linie zwei Punkte" und zugehöriger Code 170
Abbildung 6.21: Darzustellende Texte der PPNEU-Menus (Variablen „m1“ bis „mx“) 171
Abbildung 6.22: Menubelegung für das "Erstellen"-Menu von PPNEU 171