Inhaltsverzeichnis

1 Einleitung und Überblick 1

2 Das Phasenmodell für die Softwareentwicklung 3

2.1 Anforderungsermittlung 3

2.2 Systemgestaltung 5

2.3 Softwareentwurf 7

2.4 Programmierung 9

2.5 Funktions- und Leistungsüberprüfung 10

2.6 Installation 11

2.7 Wartung 12

3 Das zyklische Projektmodell 13

3.1 Produktbezogene Aufgaben 14

3.2 Prozeßbezogene Aufgaben 16

4 Projektspezifische Entwicklungsstrategie 18

4.1 Situative Adaption des Projektmodells 18

4.2 Prototyping 18

4.3 Koordination des Softwareentwicklungsprozesses 21

5 Projektorganisation und prozeßbezogene Aufgaben 23

5.1 Konstruktive Qualitätssicherung 24

5.2 Systematischer Test 25

5.3 Produktverwaltung 28

6 Projektbegleitende Dokumenation 30

7 Gestaltbildende Projekttechniken 32

7.1 Perspektive 32

7.2 Grundlagen der Zusammenarbeit legen und erneuern 33

7.3 Eine Projektsprache herausbilden 33

7.4 Rollen definieren, wahrnehmen und austauschen 34

7.5 Einbezogenheit und Vernetzung realisieren 34

7.6 Konstruktive Kritik geben und annehmen 35

7.7 Milieu zur Zusammenarbeit bilden und lebendig erhalten 35

Index i

1 Einleitung und Überblick

Diese Unterlage befaßt sich mit Projekten zur Softwareentwicklung. Sie bezieht sich auf das im Rahmen von STEPS entwickelte Verständnis von Projekten[1] und unterstützt eine sinnvolle Vorgehensweise im Projekt durch *Projekttechniken* für die Planung und Koordination des Entwicklungsprozesses und die Sicherung der Qualität der entstehenden Produkte. Als Handbuch strebt sie kurze Anleitungen an und verzichtet auf Literatur und ausführliche Begründungen.

Ein Projekt beruht auf der Zusammenarbeit von Entwicklern[2] miteinander und auf der Zusammenarbeit von Entwicklern mit Benutzern vor dem Hintergrund eines festen Entwicklungsauftrags. Dabei verdeutlichen komplementäre Sichtweisen zwei sich gegenseitig bedingende Dimensionen der Softwareentwicklung:

- Die *produktorientierte Sichtweise* betrachtet Software als eigenständiges Produkt bestehend aus Programmen und definierenden Dokumenten. Sie trennt die Herstellung vom Einsatz und versteht unter Softwareentwicklung die Herstellung des Produktes.
- Die *prozeßorientierte Sichtweise* betrachtet Software im Kontext von Lern-, Arbeits- und Kommunikationsprozessen, aus denen bei der Herstellung das Produkt Software hervorgeht, bzw. in die es beim Einsatz eingepaßt wird. Sie betrachtet die Herstellung und den Einsatz von Software als miteinander verschränkt.

Die beiden Sichtweisen spiegeln sich in *produkt-* und *prozeßbezogenen Aufgaben*, die im Projekt miteinander verbunden werden müssen. Die Projekttechniken unterstützen ihr *projektspezifisches* Zusammenspiel.

In Abschnitt 2 wird das *Phasenmodell* dargestellt. Es beschreibt die produktbezogene Arbeit der Entwickler, indem es *Entwicklungsschritte* und ihre *Ergebnisse* benennt und logisch ordnet. Zu jedem Entwicklungsschritt werden Qualitätsanforderungen, Methoden sowie die zu erarbeitenden Ergebnisse angegeben.

Zur *zeitlichen Abwicklung* dient das in Abschnitt 3 dargestellte *zyklische Modell*. Es bezieht sich auf die produkt- und die prozeßbezogene Arbeit von Entwicklern und Be-

---

[1] Die vorliegende Fassung des Projekthandbuchs wurde an der Technischen Universität Berlin im Sommersemester 1991 erstellt. Zugrundegelegt werden die *Arbeitsunterlagen zur Lehrveranstaltung Einführung in die Softwaretechnik*, C. Floyd (aktuelle Version: Universität Hamburg, 1991).

[2] Mit *Entwickler*, *Benutzer* und ähnlichen Begriffen sind im folgenden funktionelle Rollen gemeint, die von Personen beiderlei Geschlechts ausgefüllt werden können.

nutzern. Es betrachtet das Produkt als Folge von *Systemversionen* und den Prozeß als Folge von *Zyklen* aus *Herstellung* und *Einsatz* und faßt die Entwicklungsschritte des Phasenmodells versionsbezogen in größere Einheiten zusammen. So ergibt sich ein Rahmen für eine Klasse von projektspezifischen Vorgehensweisen.

Abschnitt 4 befaßt sich mit *Entwicklungsstrategien* für individuelle Projekte. Das Projektmodell muß durch die Wahl der Zyklen und Systemversionen *situativ adaptiert* und durch *exploratives*, *experimentelles* und *evolutionäres Prototyping* angereichert werden. Die Koordination der Zusammenarbeit erfolgt aufgrund eines *Projektplans* durch die *Projekt-* bzw. *Revisionsetablierung* zu Beginn jedes Zyklus und durch vorgegebene *Meilensteine* sowie aus dem Entwicklungsprozeß hervorgehende *Referenzlinien* im laufenden Zyklus.

Abschnitt 5 beschreibt übergeordnete prozeßbezogene Aufgaben bei der Softwareentwicklung. *Konstruktive Qualitätssicherung* gewährleistet die Einhaltung von Qualitätsstandards durch die Überprüfung von Zwischenprodukten und die Etablierung von Konventionen. *Produktverwaltung* ermöglicht den zuverlässigen Umgang mit der Gesamtheit der Komponenten bei Herstellung und Weiterentwicklung. *Systematischer Test* besteht in der Integration und der Validation des Produkts.

In Abschnitt 6 geht es um die projektbegleitende *Dokumentation*. Die Produktdokumentationbesteht aus den definierenden Dokumenten einschließlich Kommentaren und Begründungen und aus dem Benutzerhandbuch. Die Prozeßdokumentationhält die Historie der Entwicklung in Form von Protokollen und Tagebüchern fest.

Abschnitt 7 gibt schließlich Empfehlungen zur *Projektgestaltung*, um die kooperative Arbeit und ein gemeinsames ganzheitliche Verständnis des erwünschten Produktes, seiner Einpassung in den Einsatzkontext und des Entwicklungsprozesses selbst zu fördern.

2 Das Phasenmodell für die Softwareentwicklung

Das Phasenmodellbeschreibt die produktbezogene Arbeit der Entwickler bei der Softwareherstellung als lineare Folge von aufeinander aufbauenden Phasen und daraus resultierenden Dokumenten (siehe Abb. 1). Es *benennt* und *ordnet* Entwicklungsschritte und ihre Ergebnisse und zeigt deren *logische Abhängigkeiten*. Wir verbinden damit nicht eine Empfehlung zur zeitlichen Abwicklung von Projekten.

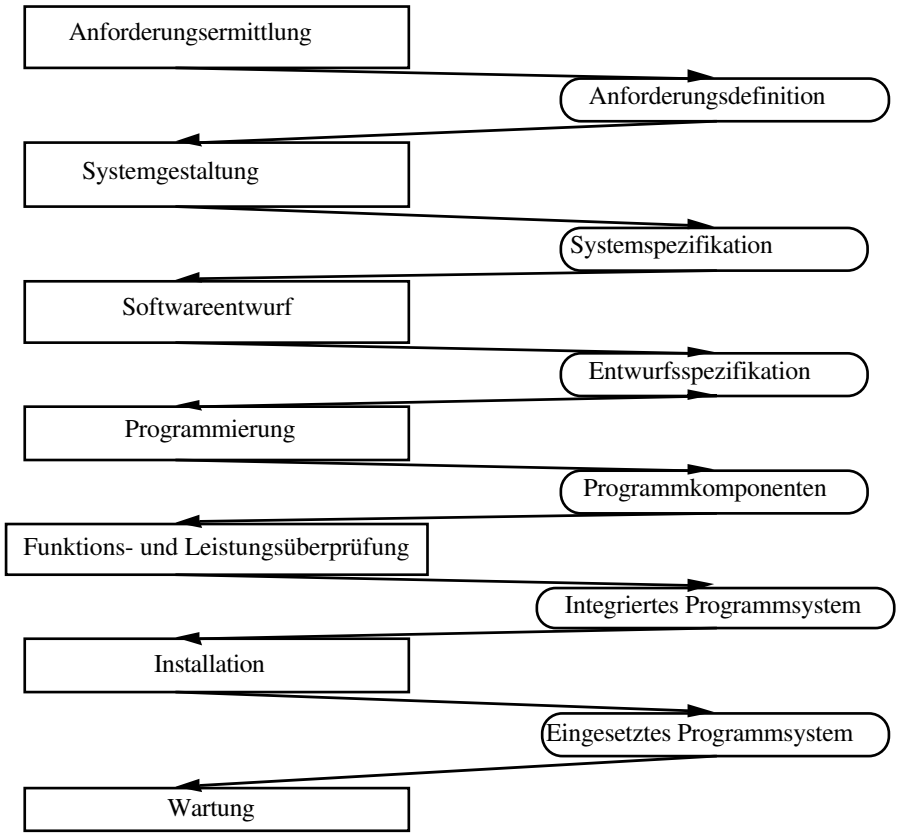

Abb. 1: Phasenmodell für die Softwareentwicklung

Im folgenden werden die Phasen und die resultierenden Dokumente kurz beschrieben. Für jede Phase werden relevante Methoden und die maßgeblichen Qualitätsanforderungen benannt. Für jedes Dokument werden die geforderten Inhalte angegeben.

2.1 Anforderungsermittlung

Diese Phase dient zur Festlegung von *Anforderungen* an die Software. Die Anforderungen stammen aus der Welt der Anwendung, dem Einsatzumfeld der Software. Dient

die Software der Unterstützung von Arbeitsprozessen, so müssen die Anforderungen aus den Erfordernissen der Arbeitsprozesse abgeleitet und aus einer möglichst bewußt eingenommenen Benutzungsperspektive ermittelt werden. Beim Einsatz dient die Software als Arbeitsmittel oder als Kommunikationsmedium für die Benutzer.

Dabei unterscheiden sich Benutzer sowohl nach ihren Aufgaben (*funktionellen Rollen*), ihrem Fachwissen und ihrer Qualifikation als auch nach ihren persönlichen und kollektiven Interessen. Da Anforderungen die Benutzungsperspektive reflektieren, können sie sich aus der Sicht einzelner Benutzer oder aus der Sicht von Benutzergruppen unterschiedlich stellen. Sowohl Lücken und Widersprüche als auch unerfüllbare Wünsche sind denkbar. Anforderungen sind nicht "gegeben". Sie werden vielmehr in einem komplexen sozialen Prozeß erarbeitet, ausgewählt und anerkannt.

Bei der Festlegung von automatisierungsfähigen und -würdigen Anteilen der Arbeitsprozesse ist ein sinnvolles Zusammenwirken von Menschen mit dem Computer zu beachten. *Automatisierungsfähig* sind nur solche Anteile der Arbeit, über die gesichertes Wissen so vorliegt, daß die maßgeblichen Einzelschritte, ihre Abhängigkeiten und ihre Ergebnisse durch Regeln beschrieben werden können. *Automatisierungswürdig*keit ist nach mehreren Kriterien zu beurteilen: nach ökonomischen im Sinne eines vertretbaren Kosten-Nutzen-Verhältnisses, nach technischen im Sinne von Machbarkeit, und nach humanen im Sinne von menschengerechter computergestützter Arbeit. Ein wesentlicher Gesichtspunkt ist die Verträglichkeit mit gesetzlichen Grundlagen.

Qualitätsanforderungen

Sie beziehen sich auf die Anliegen der Einsatzorganisation und sind bei der Bestimmung der automatisierungswürdigen Anteile der Arbeit zu beachten. Grundfragen sind:

- *Relevanz* im Einsatzkontext: Kann für die ermittelten Probleme durch den Einsatz von Computern Abhilfe geschaffen werden und welche Art von Systemkonzept erscheint sinnvoll;
- *Effektivität:* Läßt sich erwarten, daß das angestrebte System die Qualität und Produktivität der zu leistenden Arbeit steigern wird;
- *Sozialverträglichkeit*: Werden die Rechte der Benutzer oder anderer Betroffener tangiert, soziale Beziehungsgefüge geachtet und wünschenswerte Qualifikationsprofile vorgesehen;
- *Umweltverträglichkeit*: Werden mithilfe des angestrebten Systems direkt oder indirekt umweltgefährdende oder umweltschonende Produktions- und Arbeitsprozesse gefördert;

- *Arbeitserleichterung*: Kann von dem angestrebten System eine Unterstützung der Benutzer in ihren Arbeitsprozessen erwartet werden, bzw. welche neuen Tätigkeiten kommen durch den Einsatz von Computern hinzu;
- *Akzeptanzfähigkeit*: kann das angestrebte System den Betroffenen unter Berücksichtigung ihrer Aufgaben, Qualifikation, individuellen Wünsche und Interessen zugemutet werden;
- *Flexibilität*: wird das System die Fähigkeit der Einsatzorganisation, sich auf neue oder veränderte Aufgaben einzulassen, fördern oder gefährden.

Methoden

Bei den Methoden lassen sich solche zur Anforderungserhebung und zur Anforderungsbeschreibung unterscheiden:

- *Interviewtechniken* zur Unterstützung von Gesprächen mit der Anwendungsorganisation;
- Hilfsmittel zur *ablauforientierten Modellierung* von Aufgaben bzw. Benutzertätigkeiten, aufgeschlüsselt nach funktionellen Rollen oder personenbezogenen Sichten (z.B. Aufgabennetze, SA, SSA oder SADT);
- *Referenzglossare* zur Festlegung einer verbindlichen Projektsprache über den Realitätsbereich der Anwendungswelt.

Anforderungsdefinition

Die Anforderungsdefinition dient zum Festschreiben der ermittelten Anforderungen aus der Einsatzperspektive. Anforderungen sind nicht eins zu eins mit Lösungen gleichzusetzen. Die gewählte Lösung muß aufgrund der Gesamtheit der Anforderungen erarbeitet werden und das Zusammenspiel einzelner Anforderungen berücksichtigen. Nicht alle individuellen Anforderungen sind dabei umsetzungsfähig. Anforderungen lassen sich in *funktionelle, Leistungs-, Handhabungs- und Einbettungsanforderungen* klassifizieren und sind nach Benutzergruppen aufzuschlüsseln.

Die Anforderungsdefiniton kann ein eigenständiges Dokument oder ein textueller Baustein der Systemspezifkiation sein.

2.2 Systemgestaltung

Hier wird ein Lösungsvorschlag zur Erfüllung der vorliegenden Anforderungen erarbeitet. Das bedeutet ein Zusammenspiel von organisatorischen, planerischen und techni-

schen Entscheidungen, in denen die veränderten Aufgaben bzw. Tätigkeiten der Benutzer und ihre technische Unterstützung festgelegt werden und sich klärt, ob die Entwicklung von Software benötigt wird.

Soll Software entwickelt werden, so ist die *äußere Architektur* des Systems, seine *Benutzungsschnittstelle* und ein abstraktes *Modell der Objekte und Operationen* zu erarbeiten, die im Programm verankert sein sollen. Ferner sind die grundlegenden Entscheidungen über anzuschaffenden Geräte zu treffen. Für die Einbettung der Software in ihr Einsatzumfeld sind die Rahmenbedingungen der computergestützten Arbeitsprozesse zu antizipieren.

Qualitätsanforderungen

Sie beziehen sich auf die Eignung des Systems für den Einsatz und dienen bei der als Orientierung für seine Konzeption und Realisierung.

- Benutzungsorientierte Qualitätsmerkmale für interaktive Systeme sind nach DIN: Aufgabenangemessenheit, Erwartungskonformität, Steuerbarkeit, Selbstbeschreibungsfähigkeit und Fehlertoleranz.
- Betriebsorientierte Qualitätsmerkmale für einen zuverlässigen Einsatz sind: Robustheit, Ausfallsicherheit und Datenintegrität.
- Weiterentwicklungsorientierte Qualitätsmerkmale sind die problemorientierte Änderbarkeit (z.B. Adaptierbarkeit für das Einsatzumfeld oder für individuelle Benutzer) und die technische Änderbarkeit (z.B. Portabilität).

Methoden

Sie betreffen die Spezifikation der Ergebnisse der Systemgestaltung:

- Ablauforientierte Mittel zur Beschreibung der Rechnereinbettung, z.B. Aufgabennetze, SA, SSA, SADT;
- Konzepte zur Datenmodellierung, z.B. nach dem relationalen Ansatz als Entity-Relationship-Modell;
- Formale und semiformale Spezifikationstechniken für das abstrakte Modell (siehe Abschnitt 2.3 über Softwareentwurf);
- Semiformale Mittel zur Beschreibung der Benutzungsschnittstelle, z.B. Syntaxdiagramme, Backus-Naur-Form oder Interaktionsnetze.

Systemspezifikation

Dieses Dokument bildet die gemeinsame Grundlage für die Herstellung der Software und die Vorbereitung ihres Einsatzes. Es muß daher sowohl für die Entwickler eindeutige, vollständige und widerspruchsfreie Vorgaben liefern, als auch für die Benutzer die geplante Einbettung des Systems in das Einsatzumfeld in brauchbarer Weise aufzeigen.

Die Systemspezifikation enthält folgende Teile:

- Benutzermaschine, bestehend aus dem abstrakten Modell und der Benutzungsschnittstelle,
- (Klasse von) Basismaschine(n), bestehend aus vorausgesetzter Hardware und Basissoftware,
- Rechtfertigung der gewählten Lösung in Bezug auf die ermittelten Anforderungen,
- eventuelle Vorgaben für den nachfolgenden Softwareentwurf und die Programmierung.
- Angaben über die veränderten Aufgaben, die Umverteilung von Rollen und Zuständigkeiten, sowie die Einpassung des DV-Systems in die neugestalteten Arbeitsprozesse.

2.3 Softwareentwurf

Gegenstand des Softwareentwurfs ist die Erarbeitung der *inneren Architektur* des Softwaresystems. Das bedeutet die Zerlegung in geeignete Entwurfskomponenten und die Festlegung der Verknüpfung zwischen den Komponenten unter Berücksichtigung vereinbarter Kriterien. Die gewählte Zerlegung wird sowohl anhand der Vorgaben aus der Systemspezifikation als auch anhand der Entwurfskriterien überprüft und in der Regel mehrfach verfeinert und revidiert. Ist eine stabile Zerlegung erarbeitet, so müssen die Entwurfskomponenten ausspezifiziert werden.

Vorrangige Anliegen beim Softwareentwurf sind die *Bewältigung der Komplexität* des Systems sowie die Vorbereitung einer *arbeitsteiligen Programmierung* auf der Grundlage der Zerlegung.

Qualitätsanforderungen

Sie sind zu berücksichtigen bei der Wahl der Komponenten und ihrer Verknüpfungen sowie bei der Ausgestaltung der einzelnen Komponenten. Gefordert sind:

- Verständlichkeit oder intellektuelle Beherrschbarkeit ,

- Strukturelle Änderbarkeit , das heißt die Möglichkeit, Fehler zu beheben, die Effizienz zu steigern oder veränderten Anforderungen gerecht zu werden (Wartbarkeit) und
- Wiederverwendbarkeit der Komponenten als Bausteine in anderen Softwaresystemen.

Methoden

Die Methoden betreffen die Erarbeitung der Zerlegung und die Festschreibung des Ergebnisses. Dazu sind eine Fülle unterschiedlicher Konzepte erarbeitet worden, deren Eignung vom jeweiligen Problem, dem Vorwissen der Entwickler, der Lebensdauer und dem anvisierten Verbreitungsgrad der Software abhängt.

Für die praktische Systementwicklung ist eine Zerlegung in *Moduln*, die durch *Schnittstellen* verknüpft sind, von besonderer Bedeutung. Kriterien für den modularen Entwurf sind: *Lokalität* von Entwurfsentscheidungen (Geheimnisprinzip), die Bildung *prozeduraler Schnittstellen*, *Minimalität* der Schnittstellen, *Benutzt-Hierarchie* und *stufenweise Abstraktion* auf der Gesamtheit der Moduln sowie *Ökonomie* des Steuer- und Datenflusses.

Wichtige semiformale und formale Spezifikationstechniken sind:

- für einzelne Operationen: *Struktogramme* (oder Nassi-Shneiderman-Diagramme) bzw. *Pseudocode* zur Beschreibung von strukturierten Algorithmen;
- für die Auswahl von Operationen aufgrund von Entscheidungen in Abhängigkeit von Eingabe und Systemzustand: *Tabellen*, *Endliche Automaten*, *Entscheidungstabellen*;
- für die Modellierung von Daten und das datenspezifische Zusammenwirken von Operationen: *Abstrakte Datentypen*.

Entwurfsspezifikation

Dieses Dokument hält die Ergebnisse des Softwareentwurfs fest und liefert die Grundlage für die arbeitsteilige Programmierung. Es besteht aus folgenden Teilen:

- Graphische Darstellung der Zerlegung in Entwurfskomponenten und ihrer Verknüpfung (z.B. Modulstrukturdiagramm),
- Festlegung der Verknüpfungen (z.B. Schnittstellenbibliothek),

- Spezifikation der einzelnen Entwurfskomponenten mit formalen oder teilweise formalisierten Mitteln
- begründende Argumentation in Prosa zur Rechtfertigung der Entwurfsentscheidungen.

2.4 Programmierung

Die Programmierung eines größeren Softwaresystems erfolgt arbeitsteilig und basiert auf den Vorgaben der Entwurfsspezifikation. Zwischen Softwareentwurf und Programmierung besteht keine scharfe Trennung, da auch bei der Programmierung im kleinen der Entwurf weiter geführt wird. Häufig ergeben sich ferner aus der Programmierung vertiefte Einsichten, die eine Änderung der Entwurfsspezifikation erfordern. Die Programmkomponenten brauchen den Entwurfskomponenten nicht unbedingt eins zu eins entsprechen. Die Entwurfskomponenten können auch in einer Projektbibliothek verwaltet, und die Programmkomponenten durch Maßnahmen wie Overlaybildung, Makros oder den Einsatz eines Präprozessors aus ihnen abgeleitet werden.

Wichtige Probleme bei der Programmierung sind die Koordination der Fertigstellung einzelner Komponenten und die Verwaltung der dabei entstehenden Komponentenversionen. Dies erfordert ein *Konzept zur Versions- und Produktverwaltung*. In der Regel wird dieses durch eine am Rechner geführte Projektbibliothek abgesichert.

Zur arbeitsteiligen Programmierung gehört auch die Validation der einzelnen Komponenten durch *Verifikation* oder *Komponententest*: die Gewährleistung der syntaktischen Korrektheit der individuellen Programmkomponenten und die Überprüfung ihrer Arbeitsweise *anhand der Entwurfsspezifikation*.

Methoden

Methoden bei der Programmierung betreffen die systematische Entwicklung von Programmkomponenten und den Komponententest:

- Strukturierte Programmierung,
- Zusicherung über die Wirkungsweise von Algorithmen,
- Datenabstraktion mithilfe von Typen,
- Systematisches Testen von Programmpfaden ("White-Box-Test").

Qualitätsanforderungen

- *Korrektheit* der Programme bezogen auf die Entwurfsspezifikation;
- *Laufzeiteffizienz* und *Speicherökonomie* des Programmsystems je nach Leistungsanforderungen und Basismaschine;
- *Lesbarkeit* der Programmtexte durch Layout, kanonische Form, Programmierkonventionen und vereinheitlichte Schnittstellen.

Programmkomponenten

Die Programmkomponenten sind lauffähige Quellprogramme in der gewählten Programmiersprache einschließlich ihrer Dokumentation und der zugehörigen Validationsunterlagen. Sie beziehen sich auf ihre technische *Entwicklungsumgebung*: auf die Basismaschine einschließlich Betriebssystem, Datenbank, Programmiersprache und Laufzeitsystem, auf Werkzeuge, die bei der Softwareentwicklung verwendet werden, und auf Testrahmen, die als Grundlage für die individuelle Validation der Komponenten sowie für die spätere Integration des Systems dienen.

2.5 Funktions- und Leistungsüberprüfung

Um die korrekte und effiziente Funktionsweise des integrierten Programmsystems zu gewährleisten, besteht diese Phase aus Validation und Optimierung.

Die Validation erfolgt in der Regel durch Testen. Grundlage ist dabei die Systemspezifikation. Im Integrationstest wird das gesamte Programmsystem zur Ausführung gebracht und im Systemtest anhand von systematisch ausgewählten Testdaten erprobt. Diese beiden Testarten sind in der Regel nicht scharf getrennt.

Die Hauptprobleme beim Integrationstest sind die Lokalisierung von Fehlern in den einzelnen Komponenten sowie die Bereinigung von Mißverständnissen über Schnittstellen. Dazu kommt noch die Versionsverwaltung aufgrund von sukzessiven Fehlerbereinigungen und Revisionen in den Komponenten.

Der Integrationstest erfordert ein sorgfältig geplantes Vorgehen nach einer *Teststrategie* zur schrittweisen Integration der Programmkomponenten und eine darauf abgestimmte *Testorganisation* einzelner Testschritte mit definierten Teilzielen. In der Regel kommen Bibliotheken zur Produktverwaltung sowie Testwerkzeuge zum Einsatz.

Die *systematische Auswahl von Testdaten* erfolgt aufgrund des "Black-Box-Prinzips" anhand der Systemspezifikation. Zu achten ist auf eine gut begründete Überdeckung

einzelner Eingabefälle sowie auf die Berücksichtigung relevanter Kombinationen von einzelnen Eingabefällen.

Die *Optimierung* erfolgt im Bedarfsfall aufgrund von Messungen und möglichst lokaler Änderung der erarbeiteten Programme.

Integriertes Programmsystem

Das integrierte Programmsystem besteht aus der Gesamtheit der validierten Programmkomponenten einschließlich Testunterlagen und Dokumentation in der technischen Entwicklungsumgebung. Validation gewährleistet *niemals vollständige Korrektheit*. Testen kann nur die Anwesenheit von Fehlern aufzeigen, nicht deren Abwesenheit beweisen, ferner sind erschöpfende Tests bei größeren Programmen nicht möglich. Formale Korrektheitsbeweise sind in der Praxis nur begrenzt einsetzbar und zudem auch fehleranfällig. Letztlich findet Validation bezogen auf definierende Dokumente statt, die selbst Fehler enthalten können. Dazu kommen noch Mißverständnisse bezüglich der Anforderungen.

2.6 Installation

Hier ist die Einführung des entwickelten Programmsystems in sein Einsatzumfeld gemeint. Dazu gehört die Übertragung der Software aus der technischen Entwicklungsumgebung in die technische Einsatzumgebung, die Vorbereitung des Einsatzes durch organisatorische Umstellungen und die Schulung der Benutzer in der Handhabung des Systems.

Nach der Validation der Software durch die Entwickler folgt jetzt die *Evaluation* (Auswertung) des Systems in den Arbeitsprozessen der Benutzer. Die wesentlichen Probleme dieser Phase sind das Sichtbarwerden von Mißverständnissen oder Lücken in Bezug auf die Anforderungen, die zu großen Akzeptanzschwierigkeiten führen und Anlaß zu kostspieliger Reparatur und Weiterentwicklung geben können. Diese Probleme lassen sich nur durch einen bewußt gestalteten projektbegleitenden Kommunikationsprozeß zwischen Entwicklern und Benutzern auffangen, bei dem rechtzeitig Rückkopplung in das laufende Entwicklungsprojekt erfolgt.

Eingesetztes Programmsystem

Das eingesetzte Programmsystem existiert in der jeweils gültigen Systemversion, die die erwünschte Funktionalität annähert, in der Regel Fehler enthält, sowie Erweiterungen und Qualitätsverbesserungen nahelegt.

Durch die Evaluation im Einsatz werden die Anforderungen konkretisiert, und neue Anforderungen erzeugt. Zusätzlich befindet sich sowohl die technische Ressourcenwelt (Geräte, Basissoftware) als auch das Einsatzumfeld der Software im ständigen Wandel. Was das eingesetzte Programmsystem für einzelne bedeutet, hängt sowohl von der jeweils eingesetzten Systemversion als auch von der darauf aufbauenden Arbeitsorganisation am jeweiligen Einsatzort und letztlich von den Kenntnissen der Benutzer selbst ab. So entwickeln sich spezifische Benutzungstraditionen im Rahmen von Arbeitsprozessen.

2.7 Wartung

Software-"Wartung" ist ein gebräuchlicher, aber irreführender Begriff, da Software keinen Verschleiß kennt. Wartung bedeutet hier Fehlerbereinigung ("Pflege") und Weiterentwicklung. Zwischen diesen beiden Aufgabenbereichen gibt es allerdings keine scharfe Trennung. Zum einen ist häufig nicht klar, was als "Fehler" anzusehen ist, was ein Mißverständnis war, und was bereits eine neue Anforderung bedeutet. Zum anderen kann die Fehlerbereinigung Anlaß zur Weiterentwicklung werden.

Pragmatisch läßt sich zwischen der *Pflege der aktuellen Version*, bei der kleinere Fehler beseitigt und begrenzte Weiterentwicklungen vorgenommen werden, und der *Entwicklung einer neuen Version* zur Fehlerbereinigung, Qualitätsverbesserung sowie zur Berücksichtigung neuer Anforderungen unterscheiden.

Bei mehrfach eingesetzten Produkten überlagert sich die Bildung von *zeitlich aufeinanderfolgenden Versionen* und die parallel existierenden *einsatzspezifischen Varianten*. Eine Systemversion setzt sich dann aus der *Konfiguration* jeweils gültiger Versionen und Varianten von Programmkomponenten zusammen.

3 Das zyklische Projektmodell

Als Grundlage zur *zeitlichen Projektabwicklung* dient ein zyklisches Projektmodell, siehe Abb.2. Die Systementwicklung wird hier als Folge von Entwicklungszyklen angesehen. Jeder Zyklus hat die Herstellung und den Einsatz einer Systemversion zum Gegenstand. Das Projektmodell reflektiert die Sicht der Entwickler auf die Systementwicklung ebenso wie die Sicht der Benutzer und weist die gemeinsamen und gesonderten Aufgaben beider Gruppen aus.

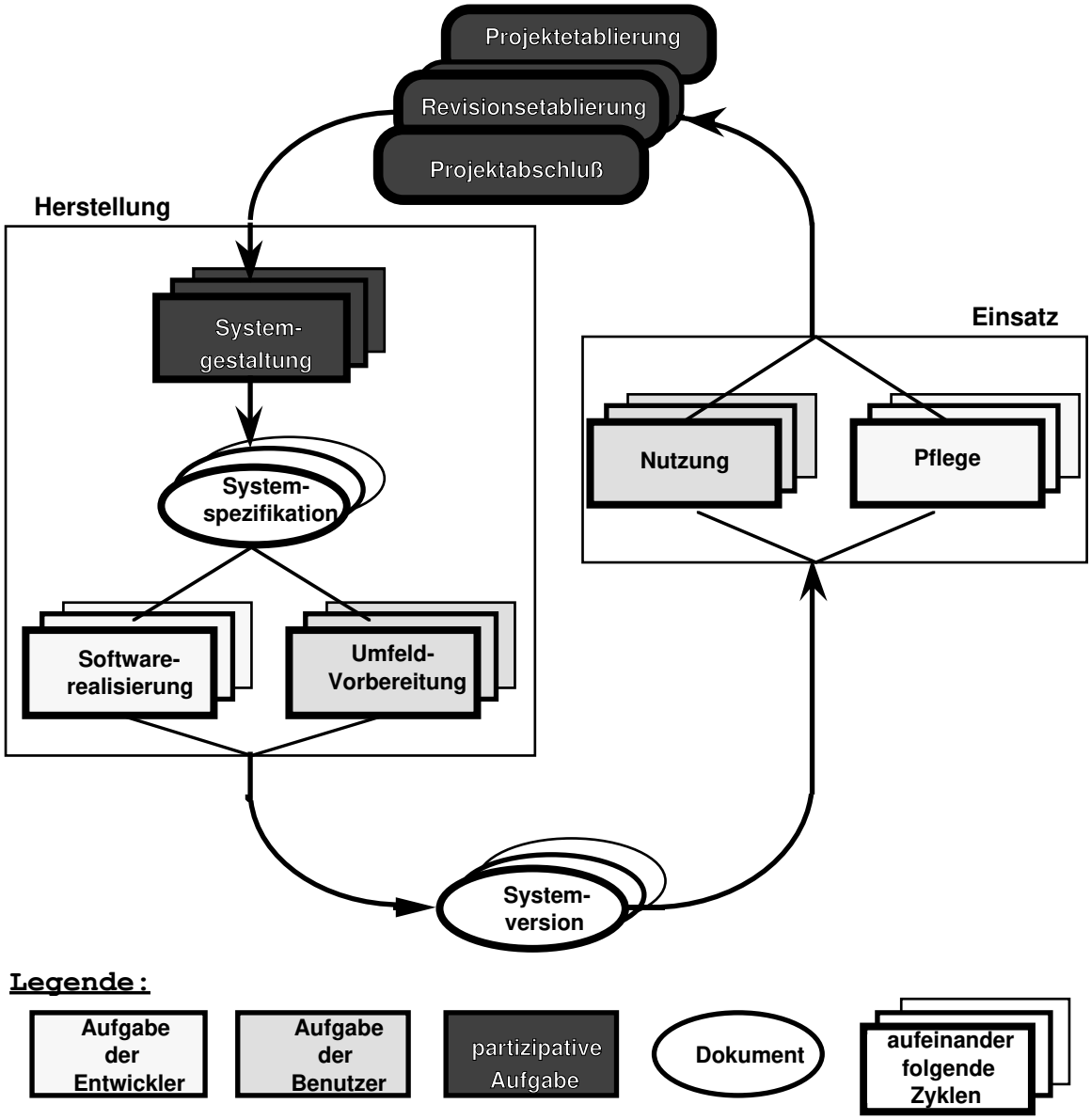

Abb. 2: Zyklisches Projektmodell

Das zyklische Projektmodell modelliert eine Klasse von möglichen Vorgehensweisen und liefert einen Bezugsrahmen für die Festlegung einer projektspezifischen Vorgehensweise durch situative Anpassung (siehe Abschnitt 4). Es verdeutlicht das Zusammenspiel von produktbezogener und prozeßbezogener Arbeit bei der Softwareentwicklung. Die produktbezogenen Aufgaben leiten sich aus den im Phasenmodell benannten Entwicklungsschritten ab. Der Zusammenhang zwischen dem Phasenmodell und dem zyklischen Modell ist durch folgende Aspekte gegeben:

- Das zyklische Modell versteht unter einem Produkt *eine Folge von Systemversionen*. Im Gegensatz zu Phasen für die Herstellung des gesamten Produkts beziehen sich sämtliche Entwicklungsschritte und ihre Ergebnisse auf die jeweilige Version und bauen inkrementell auf die vorangegangenen Versionen auf.

- Die Phasen werden zyklisch verschränkt und zu größeren Einheiten (*Systemgestaltung, Softwarerealisierung*), gruppiert, die komponentenweise und zeitlich flexibel fein strukturiert werden können. Dies begünstigt auch die Einbeziehung von *Prototyping* in den Entwicklungsprozeß, siehe Abschnitt 5.

- Die im Phasenmodell verankerte *Wartung* wird in die *Pflege* der aktuellen Version und die *Weiterentwicklung* des Systems aufgelöst. Die Weiterentwicklung erfolgt nach der Revision als Herstellung einer neuen Version im nächsten Zyklus.

3.1 Produktbezogene Aufgaben

Die*Systemgestaltung* integriert die Phasen Anforderungsermittlung und Systemgestaltung des Phasenmodells (siehe Abb. 3). Dabei kooperieren nach Möglichkeit Benutzer und Entwickler. Die Systemgestaltung vollzieht sich in Zyklen der Analyse, Synthese und Revision. In wechselseitigen Kommunikations- und Lernprozessen entsteht das zur Bestimmung der Eigenschaften des Systems benötigte neue Wissen und die zur Bewertung seiner Qualität erforderliche gemeinsame Kompetenz. So erhält das System allmählich seine Konturen und seine stabile Gestalt.

Technisch können diese Prozesse durch exploratives Prototyping bei der Ermittlung von Anforderungen oder durch experimentelles Prototyping (z.B. bei der Gestaltung der Mensch-Rechner-Interaktion) unterstützt werden (vgl. Abschnitt 5).

Die *Systemspezifikation* entspricht dem gleichnamigen Dokument im Phasenmodell. Sie enthält auch die Vorgaben für die in der Einsatzorganisation durchzuführenden Maßnahmen zur Qualifikation und Beschaffung sowie für organisatorische und räumliche Umstellungen.

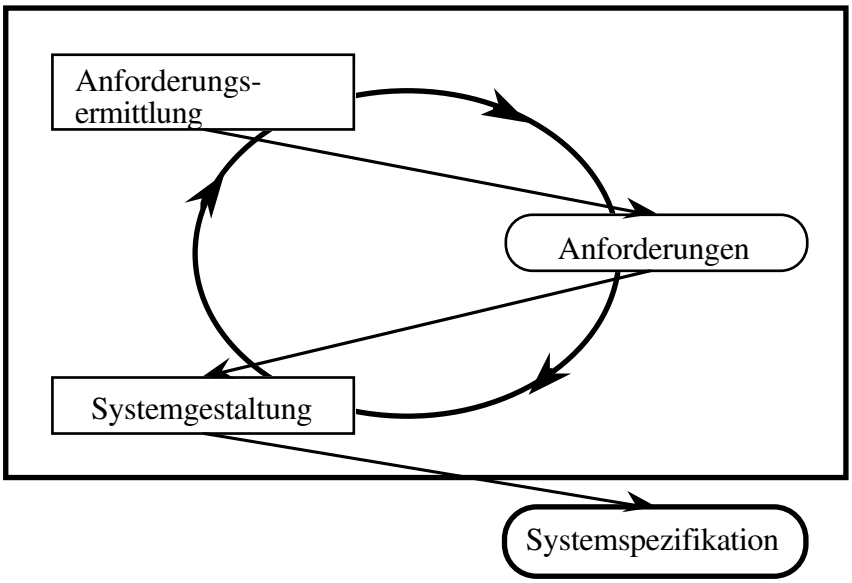

Abb. 3: Zusammenhang zwischen Phasenmodell und zyklischem Modell: Systemgestaltung

Die *Softwarerealisierung* wird von den Entwicklern durchgeführt. Sie umfaßt die Aufgaben: Softwareentwurf, Programmierung und Funktions- und Leistungsüberprüfung aus dem Phasenmodell (siehe Abb. 4). Auch hierbei vollziehen sich Zyklen von Analyse, Synthese und Revision, die durch experimentelles Prototyping und Ausbaustufenbildung unterstützt werden können (siehe Abschnitt 5).

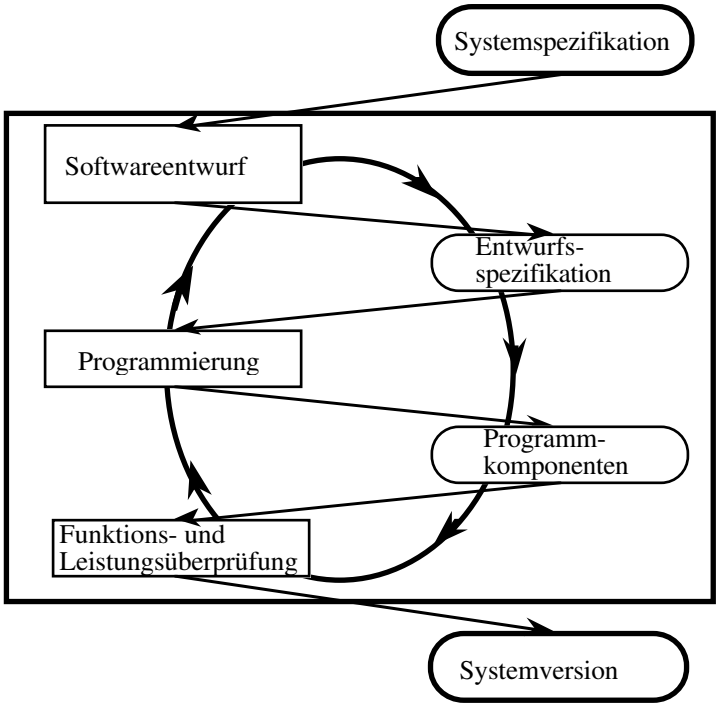

Abb. 4: Zusammenhang zwischen Phasenmodell und zyklischem Modell: Softwarerealisierung

Die *Umfeldvorbereitung* wird von den Benutzern durchgeführt. Sie beinhaltet organisatorische, planerische und räumliche Maßnahmen, die im Hinblick auf den Einsatz erforderlich sind. Sie schließt Schulungs- und Qualifikationsmaßnahmen für die betroffenen Mitarbeiter ein und führt zur Erarbeitung von Auswertungsgrundlagen für die jeweilige Systemversion.

Zur Bildung der *Systemversion* ist nach dem Phasenmodell die Installation des integrierten Programmsystems erforderlich. In die Systemversion gehen ein: der Einsatzrechner und das Softwaresystem ebenso wie alle Dokumente, die von den Benutzern (Auswertungsunterlagen) und den Entwicklern (Handbücher, Beschreibung der Einschränkungen gegenüber der Systemspezifikation) erstellt werden und als Unterlagen für die Benutzung und Pflege sowie für Revisionen des Systems dienen.

Die *Nutzung* der Version ergibt sich aus den regulären Arbeitsprozessen der Einsatzorganisation. Hier kooperieren in der Regel ausschließlich die Benutzer miteinander. Die *Pflege* der aktuellen Systemversion wird von den Entwicklern durchgeführt. Sie besteht in der Fehlerbehebung, der Adaption für die Nutzung und der Optimierung.

3.2 Prozeßbezogene Aufgaben

Die prozeßbezogenen Aufgaben dienen zur *Etablierung* und *Koordination des Softwareentwicklungsprozesses*. Im Modell ist nur die Projekt- bzw. Revisionsetablierung explizit aufgeführt. In Abschnitt 43 werden die Konzepte zur Prozeßkoordination näher beschrieben.

Bereits vor dem eigentlichen Beginn des Projekts muß geklärt sein, welche Aufgabenbereiche durch das zu entwickelnde System unterstützt und welche Anliegen umgesetzt werden sollen. Insbesondere müssen die am Projekt Beteiligten bereit sein, sich auf kooperative Arbeitsprozesse zur Entwicklung des Systems einzulassen und sich auf eine gemeinsame Vorgehensweise und die hiermit verbundenen Verpflichtungen zu einigen. Ein allen Beteiligten gemeinsames Verständnis vom Entwicklungsprojekt und seinen Zielen ist die zentrale Voraussetzung für die Durchführung der Systementwicklung.

Der erste Entwicklungszyklus wird durch die von den Entwicklern und Benutzern gemeinsam vollzogene *Projektetablierung* initiiert. Sie dient zur koperativen Erarbeitung des Projektplans sowie zur Verständigung über die Art der im Projekt geplanten Zusammenarbeit.

Das Ergebnis der Projektetablierung ist ein grobes *Systemkonzept*, aus dem sich die produktbezogenen Aktivitäten ergeben, sowie ein *Projektplan*, aus dem sich die prozeßbezogenen Aktivitäten herleiten.

Im *Erstzyklus* ist die Erstellung und der Einsatz einer Erstversion vorgesehen. Wie alle weiteren Versionen muß die Erstversion einsatzfähig sein und die Arbeitsprozesse der Benutzer sinnvoll unterstützen.

Jeder nachfolgende *Revisionszyklus* wird durch eine Revisionsetablierung initiiert, bezieht sowohl softwareinterne Gesichtspunkte als auch die vereinbarten revidierten Anforderungen ein und hat die Herstellung sowie den anschließenden Einsatz einer neuen Version zum Gegenstand.

Bei der *Revisionsetablierung* wird ähnlich wie bei der Projektetablierung von den Entwicklern und Benutzern kooperativ vereinbart, welche der neuen und veränderten Anforderungen berücksichtigt, welche Probleme gelöst werden sollen und wie die Durchführung der Revision geplant ist.

4 Projektspezifische Entwicklungsstrategie

Das zyklische Projektmodell erhebt nicht den Anspruch, eine ideale Vorgehensweise für sämtliche Projekte zu bieten. Es liefert vielmehr einen Rahmen, der zu einer Klasse von situationsspezifischen Vorgehensweisen für individuelle Projekte führt.

4.1 Situative Adaption des Projektmodells

Zur Bestimmung einer situationsspezifischen Vorgehensweise gehören

- die Festlegung der Art, der Anzahl, des Funktionsumfangs und der terminlichen Planung von Systemversionen,
- die Klärung der Art und des Ausmaßes an Benutzerbeteiligung sowie die Maßnahmen zur Freisetzung und Qualifikation der Benutzer,
- die Erarbeitung von Grundlagen für die Auswertung von Systemversionen, die Rückkopplung in das laufende Projekt und die schrittweise Umstellung der Einsatzorganisation auf die computergestützten Arbeitsprozesse vorzuplanen.

Die gewählte Vorgehensweise findet ihren Ausdruck im bereits oben erwähnten *Projektplan* und schlägt sich in der nachfolgend beschriebenen *Koordination des Entwicklungsprozesses* nieder.

4.2 Prototyping als Teil einer Entwicklungsstrategie

Prototyping ist ein technisches Verfahren zur Unterstützung von kooperativen Erkenntnisprozessen bei der Softwareentwicklung anhand von ausführbaren Vorversionen des in der Entwicklung befindlichen Produkts. Prototyping kann die Kooperation zwischen Entwicklern und Benutzern aber auch die Kooperation im Entwicklerteam unterstützen. Bei der Wahl einer Prototyping-Strategie muß die vorliegende Basismaschine (ggf. aufzufächern in Entwicklungs- und Einsatzumgebung), die Lebensdauer des zu entwickelnden Systems und das im Einzelfall vorliegende Verhältnis zwischen Entwicklern und Auftraggebern/Benutzern bis hin zur Vertragsgestaltung berücksichtigt werden.

Festzulegen sind die Art und die Anzahl der geplanten Prototypen in ihrem Verhältnis zum angestrebten Produkt sowie die Rückführung der Ergebnisse in den weiteren Entwicklungsprozeß.

Je nach dem maßgeblichen Anliegen lassen sich exploratives, experimentelles und evolutionäres Prototyping unterscheiden.

Entwicklungsschritte beim Prototyping

Prototyping besteht selbst aus vier Schritten: *Funktionsauswahl*, *Konstruktion*, *Auswertung* und *Weiterverwendung* des Prototypen.

Bezüglich der Funktionswahl gibt es die zwei Grundausprägungen *horizontales Prototyping*, bei dem alle anvisierten Systemfunktionen in vereinfachter Form realisiert werden, und *vertikales Prototyping*, bei dem nur ein Teil der Systemfunktionen in der geplanten Form realisiert werden. Häufig wird eine Mischung dieser beiden Konzepte verwendet.

Die Konstruktion erfolgt entweder mit speziellen Mitteln (Sprachen oder Anwendungsgeneratoren) oder mit denselben Mitteln wie die Produktentwicklung. Die technischen Möglichkeiten haben einen wesentlichen Einfluß auf den Aufwand bei der Herstellung und auf den Möglichkeiten zur Weiterverwendung von Prototypen.

Die Auswertung muß zwischen den beteiligten Personen aufgrund von vereinbarten Bewertungsmaßstäben sorgfältig geplant werden. Besonders sensibel ist die dabei stattfindende wechselseitige Kommunikation: die Unsicherheiten und Überraschungen beim Lernen, das Einbringen und die Annahme von Kritik, die Verständigung über Wünschenswertes und Machbares und die Verbindlichkeit des Gelernten für den weiteren Entwicklungsprozeß.

Die Weiterverwendung eines Prototypen ist nur dann möglich, wenn die Herstellung des Prototypen und des Produktes in einer technisch integrierten Umgebung erfolgt. Sie ist nur dann sinnvoll, wenn der Prototyp einsatzorientierte Qualitätsmerkmale aufweist. Prototyping kann durchaus erfolgreich sein, auch wenn der Prototyp selbst weggeworfen wird.

Exploratives Prototyping

Diese Art von Prototyping dient zum Ausloten der erwünschten Funktionalität des angestrebten Systems. Ihrem Charakter nach ist sie informell und dient dazu, die Kommunikation mit Benutzern über Anforderungen zu unterstützen sowie die Realisierbarkeit von anvisierten Systemfunktionen auf der vorliegenden Basismaschine zu klären. Exploratives Prototyping findet bei der Systemgestaltung statt und sichert die im zyklischen

Modell vorgesehene Verschränkung zwischen Systemgestaltung und der Anforderungsermittlung ab.

Experimentelles Prototyping

Diese Art von Prototyping dient zum Nachweis der Tauglichkeit von vorgeschlagenen und in der Regel bereits spezifizierten Lösungsvorschlägen. Experimentelles Prototyping wird je nach den Gegebenheiten des Projekts für unterschiedliche Zwecke verwendet:

- für die *Gestaltung der Benutzungsschnittstelle* in Kommunikation mit den Benutzern,
- für die *vollständige* oder *teilweise Simulation* des in der Spezifikation verankerten Modells,
- zur Demonstration der Einlösbarkeit von kritischen Entwurfsentscheidungen
- zur Bereitstellung eines *adaptionsfähigen Basissystems*, bei dem spezifische Benutzerwünsche je nach Einsatzkontext geklärt und ergänzt werden können, und
- zur Entwicklung eines *Programmskeletts*, bei dem jede Entwurfskomponente in minimaler Form aber verknüpft mit allen anderen realisiert wird, um die Funktionsweise des Systems im Ganzen zu erproben.

Evolutionäres Prototyping

Diese Art von Prototyping findet dort statt, wo ein inkrementelles Vorgehen bei der Softwarerealisierung gewählt wird. Dies ist beim Zusammenspiel von Softwareentwurf und Programmierung und bei der Programmierung in Ausbaustufen zu empfehlen.

Grundlage dafür ist die Systemspezifikation, aus der im Softwareentwurf eine Zerlegung in Komponenten erarbeitet und in der Entwurfsspezifikation festgehalten wird. Die Verschränkung von Softwareentwurf und Programmierung wird aufgrund einer Ausbaustufenplanung gestaltet, bei der jede Ausbaustufe eine sinnvolle Teilmenge der angestrebten Systemfunktionen anbietet.

Zuerst wird etwa eine Ausbaustufe 0 in Form eines Programmskeletts implementiert und getestet. Die Ausbaustufe 0 dient dann als Rahmen für alle weiteren Systemtests. Parallel dazu wird die Entwufsspezifikation um die zu Ausbaustufe 1 gehörenden Anteile erweitert. Die Auswahl von Systemfunktionen für die Ausbaustufe 1 und alle weiteren erfolgt nach projektspezifischen Gesichtspunkten. Anschließend wird Ausbaustufe 1 implementiert und getestet, und so weiter.

Bezogen auf jede einzelne Komponente ist das Vorgehen dabei linear: Entwurf vor Programmierung vor Test. Bezogen auf das Produkt als Ganzes ist das Vorgehen zyklisch: einzelne Ausbaustufen durchlaufen zeitlich parallel verschiedene Entwicklungsstufen.

4.3 Koordination des Softwareentwicklungsprozesses

Die Koordination der Projektarbeit erfolgt durch das Zusammenspiel von Projekt- bzw. Revisionsetablierung zu Beginn der Entwicklungszyklen mit *Meilensteinen* und *Referenzlinien* während der laufenden Zyklen.

Durch das Projektmodell sind nur zwei *Meilensteine* (Produkte und Zeitpunkte) vorgegeben: die Systemspezifikation und die Systemversion. Sie werden bei der Projektetablierung vereinbart und im Projektplan verankert. Für die Systemspezifikation gibt es ein Produktmuster, da sie als vertragliche Grundlage für den Auftraggeber und den Auftragnehmer dient und als Bezugsdokument während einer Versionsabnahme fungiert.

Zur Koordination des Entwicklungsprozesses orientieren wir uns an dem Konzept der *Referenzlinien*. Sie werden aus dem Entwicklungsprozeß heraus von den Beteiligten nach Bedarf festgelegt und können daher nicht an einer bestimmten Stelle im Projektmodell dargestellt werden.

Eine Referenzlinie definiert jeweils einen zu erreichenden Projektzustand durch das Vorliegen von Zwischenprodukten. *Zwischenprodukte* sind neben Dokumenten auch beispielsweise Glossare, Prototypen, Ausbaustufen oder Simulationen sowie Konventionen oder andere Festlegungen der gemeinsamen Arbeit. Spätestens dann, wenn dieser Projektzustand erreicht ist, muß die nächste Referenzlinie definiert sein.

Eine Referenzlinie ist benannt und besteht aus:

- einer *Liste von Zwischenprodukten*, die in einem bestimmten Zustand vorliegen müssen, sowie anderen Voraussetzungen für die weitere Arbeit;
- *Kriterien zur Bewertung*, die auf Merkmale der Zwischenprodukte, der Ressourcen oder Qualifikationen Bezug nehmen;
- *Verfahren zur Bewertung*: Wie soll das Produkt bewertet werden (z.B. durch Review, Fragebogen oder Test)? Wer soll bewerten? Wer hat oder erhält Entscheidungskompetenz?

Abb. 5 zeigt an einem Beispiel, wie ausgehend von der Projektetablierung (PE) die ersten beiden Referenzlinien (R1 und R2) von den Entwicklern und Benutzern definiert.sind. Teil der Referenzlinie; R2 ist es, die nächste(n) Referenzlinie(n) zu definieren (hier R3).

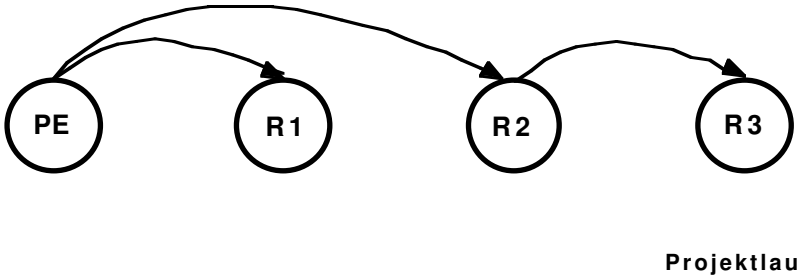

Abb. 5: Koordination eines Prozesses durch Referenzlinien

5 Projektorganisation und prozeß-bezogene Aufgaben

Eine transparente Projektorganisation beruht auf der Vereinbarung von benannten Verantwortlichkeiten und ihrer Wechselwirkung. Da die Einbettung von Projekten in übergreifende Organisationsstrukturen höchst unterschiedlich ist, betrachten wir im folgenden nur die produkt- und prozeßbezogenen Aufgaben *im Projekt*, die in funktionelle Rollen zusammengefaßt werden können. Sie sind den beteiligten Personen entweder fest zugeordnet oder werden von ihnen abwechselnd wahrgenommen.

Der *Projektleiter* trägt die fachliche und terminliche Verantwortung für das Projekt als Ganzes. Das heißt, er muß sowohl die produktbezogene als auch die prozeßbezogenen Aufgaben in ihrer Gesamtheit überblicken als auch die spezifischen Verantwortlichkeiten der einzelnen Projektmitglieder in ihrer Wechselwirkung vereinbaren und aufrechterhalten. In der Regel wird er dabei von einem *Projektassistent* unterstützt.

Die produktbezogene Arbeit wird unter den *Entwicklern* geteilt. Je nach Gegebenheit findet die Arbeitsteilung schon bei der Systemgestaltung, beim Softwareentwurf oder erst bei der Programmierung. Eine frühe Arbeitsteilung kann zum Beispiel sinnvoll sein, wenn heterogene Teile der Anwendungswelt als lose gekoppelte Subsysteme ausdifferenziert sind. Oft sind Entwickler für unterschiedliche Aspekte der Anwendung oder für bestimmte Teile der Entwicklunsumgebung besonders qualifiziert. Auf jeden Fall sind Entwickler in ihrem Bereich eigenverantwortlich und über die festgelegten Schnittstellen wie in einem Vertragsverhältnis mit den anderen Entwicklern verbunden.

Um die Perspektive der Benutzung bei der Herstellung kontinuierlich einbringen zu können, kann ein *Virtueller Benutzer* unter den Projektmitgliedern eingerichtet werden. Die Aufgabe des virtuellen Benutzers ist, bei Projektbesprechungen und -entscheidungen in Abwesenheit der Benutzer die antizipierte Sicht des Systems als Arbeitsmittel bzw. Kommunikationsmedium in den Arbeitsprozesses einzubringen. Ein virtueller Benutzer ergänzt die Kommunikation mit den echten Benutzern. Sind die echten Benutzer nicht bekannt oder nicht verfügbar, so schaftt der virtuelle benutzer bis zu einem gewissen Grad einen Ausgleich für diesen Mangel.

Zu beachten ist das projektspezifische Zusammenspiel zwischen Arbeitsteilung und gemeinsam gelingender Arbeit, dem die im folgenden beschriebenen prozeßbezogenen, übergeordneten Aufgaben sowie die in Abschnitt 7 aufgezeigten Grundzüge der Projektgestaltung dienen.

5.1 Konstruktive Qualitätssicherung

Qualität von Software wird, wie in Abschnitt 2 erläutert, durch Anforderungen oder erwünschte Merkmale charakterisiert. Diese müssen projektbezogen zwischen den Beteiligten konkretisiert, vereinbart und sichergestelltt werden. Konstruktive Qualitätssicherung besteht in Maßnahmen zur Etablierung und Gewährleistung von projektspezifischen Qualitätsmaßstäben. Diese beziehen sich auf das Produkt und seine Zwischenprodukte und müssen im Laufe des Entwicklungsprozesses entwickelt und eingelöst werden.

Da sämtliche Zwischenprodukte des Phasenmodells Dokumente sind, ist eine wesentliche Teilaufgabe der Konstruktiven Qualitätssicherung das *Überprüfen* von *Dokumenten.* Dies kann entweder auf individueller Basis in einem *Autor-Kritiker-Zyklus* durch einen Experten erfolgen, um verschiedene Sichtweisen zu kreuzen oder gemeinschaftlich in einem *Review*, um das Dokument als von allen Beteiligten getragenes Zwischenprodukt anzuerkennen und zu verabschieden. Dies muß anhand von vereinbarten Kriterien und systematisch ausgewählten Testfällen erfolgen. Die Vereinbarung von Qualitätsmaßstäben erfolgt zusammen mit der Bildung von Referenlinien im Entwicklunsprozeß.

Aufgabe ist der Qualitätssicherung ist es, die Bewertungsgrundlagen für die den Referenzlinien zugeordneten Zwischenprodukte in Interaktion mit den Entwicklern zu erarbeiten und Verfahren zu ihrer Überprüfung.festzulegen.

Die Überprüfung von Dokumenten bedeutet das Geben und Annehmen von *Konstruktiver Kritik*. Konstruktive Kritik bewertet ein vorgelegtes Dokument als (meist vorläufiges) Ergebnis eines Einzelnen oder einer Gruppe im Rahmen eines gemeinsam getragenen Erkenntnisprozesse. Sie erfolgt daher sachbezogen und besteht sowohl in einer Würdigung des bereits Erreichten als auch im Aufzeigen noch verbleibender Mängel. Sie erfordert von allen Beteiligten wechselseitige Achtung und die Bereitschaft zum gemeinsamen Suchen nach Qualität.

Ferner ist die Qualitätssicherung zuständig für die Bildung und Einhaltung von *Konventionen* zur gemeinsamen Arbeit, die wo möglich eine wünschenswerte Standardisierung und größere Transparenz mit sich bringen. Konventionen bilden eine wichtige Grundlage zur zuverlässigen arbeitsteiligen Softwareentwicklung, da sie den Umgang mit Programmen, die von anderen geschrieben wurden, erheblich erleichtern. Sie kommen in verschiedenen Bereichen zum Einsatz:

- *Programmier*konventionen regeln die Verwendung der Programmiersprache so, daß saubere Konstrukte für Algorithmen und Datenstrukturen ausgewählt oder simuliert, und fehleranfällige Sprachmittel vermieden werden;

- *Layout*-Konventionen legen die textuelle Gestaltung der Programme im Hinblick auf Lesbarkeit fest;

- *Namens*konventionen sind innerhalb von Programmkomponenten, beim Aufbau von Schnittstellen und bei allen gemeinsam benutzten Programmen und Daten zu empfehlen;

- Konventionen zur *Kommentierung* von Programmen dienen zur Vereinheitlichung der textuell begründenden Erläuterungen;

- Konventionen zur *Nutzung von Systemressourcen* helfen, Fehler beim Umgang mit dem Betriebssystem zu vermeiden.

Obwohl die Notwendigkeit von Konventionen leicht einsehbar ist, ist ihre Vereinbarung oft ein gruppendynamisch schwieriger Prozeß, da jeder ein Stück weit von seinem gewohnten Arbeitsstil abrücken muß. Daher ist stets sorgfältig abzuwägen zwischen der Möglichkeit zu individuellem Freiraum und den Vorteilen verbindlicher Form. Es sollte nicht mehr standardisiert werden, als für eine transparente Zusammenarbeit und zur Vermeidung von Fehlern und Reibungsverlusten notwendig ist.

5.2 Systematischer Test

Nach dem Phasenmodell sind verschiedene Testarten nach ihrer Stellung im Entwicklungsprozeß, nach verantwortlichen Durchführenden und maßgeblichen Validationsgrundlagen zu unterscheiden:

- Der *Komponententest* erfolgt im Rahmen der Programmierung durch die Entwickler. Er dient zur Sicherstellung der syntaktischen und Korrektheit und zur Überprüfung der Funktionsweise einzelner Komponenten bzogen auf die Entwurfsspezifikation. Er besteht in der systematischen Erprobung oder Überdeckung aller einzelnen Elemente (z.B. Anweisungen oder Fallunterscheidungen) sowie aller einzelnen Pfade von Programmen.

- Der *Integrations-* und der*Systemtest* erfolgen im Rahmen der Funktions- und Leistungsüberprüfung durch die Testgruppe. Der Integrationstest dient zur Überprüfung der vereinbarten Schnittstellen. Er ist nicht scharf trennbar vom Systemtest, der die Funktionsweise des Programmsystems auf der Grundlage der Systemspezifikation gewährleistet. Dabei wird die vereinbarte Funktionsweise der Programmkomponenten vorausgesetzt. Die Daten oder Testfälle werden durch systematische Überdeckung aller Einzeleingaben sowie relevanter Kombinationen von Eingaben anhand der Systemspezifikation festgelegt.

- Der *Abnahmetest*.erfolgt gegebenenfalls im Rahmen der Installation durch den Kunden. Er gewährleistet die erwünschte Funktionsweise des Systems in der technischen Einsatzumgebung mit den echten Daten des jeweiligen Einsatzumfeldes.

Systematischer Test bedeutet, die Gesamtheit der verschiedenen Testarten im Verlauf des Projekts zu planen, zu koordinieren und zu dokumentieren, sowie den Integrations- und Systemtest durchzuführen. Dazu sind ein Zusammenspiel von Testrategie und -organisation, die Festlegung von Testdaten sowie die Verwendung von Testwerkzeugen erforderlich.

Die *Teststrategie* betrifft die Vorgehnsweise bei der Integration des Programmsystems. Dabei ist das Zusammenspiel von Komponententest und Integrationstest sowie die Entwicklung von Ausbaustufen zu beachten. Es gibt zwei Grundformen, die auf eine hierarchische Struktur des Programmsystems (z.B. Benutzt-Struktur der Moduln) Bezug nehmen:

- Beim *Top-Down-Test* wird die hierarchisch übergeordnete Komponente (z.B. Haupt- oder Steuermodul) zuerst getestet. Sie bildet den *Systemkern*. Sämtliche untergeordneten Komponenten werden durch *Platzhalter* ersetzt, die nur die Aufrufbarkeit vom Hauptmodul aus gewährleisten. Ist die erwünschte Funktionsweise des Systemkerns etabliert, so wird ein Platzhalter durch die echte Komponente ersetzt, wobei gegebenenfalls deren Verknüpfung mit weiteren Moduln durch neue Platzhalter simuliert wird. Dadurch entsteht ein vergrößerter Systemkern. Durch Fortsetzung dieses Verfahrens wird nach und nach die erwünschte Funktionsweise des gesamten Systems gewährleistet.

- Beim *Bottom-Up-Test* werden zunächst die einzelnen Komponenten der untersten Hierarchiestufe in einem *Testrahmen* getestet, der die Aufrufe von der nächsthöheren Hierarchiestufe (z.B. vom benutzenden Modul aus) simuliert. Ist deren erwünschte Funktionsweise gewährleistet, so wird der Testrahmen durch eine (oder mehrere) übergeordnete Komponenten ersetzt, die selbst gegebenenfalls wieder in einen ihnen zugeordneten Testrahmen eingebettet werden. Dieses Verfahren wird schrittweise auf alle Komponenten ausgedehnt, bis die erwünschte Funktionsweise des Gesamtsystems erreicht ist.

Mit diesen beiden Grundformen sind Vor- und Nachteile verbunden, die bei der Bildung einer projektspezifischen Teststrategie berücksichtigt werden müssen:

- Die Möglichkeit zur *Fehlerlokalisierung* und zur einheitlichen Fehlerbehebung ist ein wesentlicher Vorteil des Top-Down-Tests, da hier immer ein abgesicherter und in sich konsistenter Systemkern vorliegt, der mit genau einer neuen Komponente verknüpft wird. Eine Gefahr beim Bottom-Up-Test ist dagegen die Herausbildung von miteinander unverträglichen Teilsystemen.

- Die Flexibilität in der *Terminplanung* bezogen auf die zeitgerechte Verfügbarkeit der benötigten Komponenten spricht eher für den Bottom-Up-Test, da hier mehr Kombinationsmöglichkeiten bestehen und ganz unterschiedliche Reihenfolgen gewählt werden können.

- Der *Zusatzaufwand* zur Bereitstellung der benötigten Platzhalter bzw. Testrahmen hängt von der gewählten Entwicklungsstrategie und der technischen Entwicklungsumgebung ab. So fallen bei der Erarbeitung eines Programmskeletts als Ausbaustufe sämtliche Platzhalter mit an, andererseits gibt es Testwerkzeuge, die die automatische Generierung von testrahmen ermöglichen.

In der Praxis wird daher meist eine Mischform von Top-Down- und Bottom-Up-Test geplant. und fortlaufend anhand der tatsächlichen Gegebenheiten des Entwicklungsprozesses angepaßt.

Die *Testorganisation* betrifft die Planung und Koordination einzelner Teiltests oder *Testschritte* zur Erreichung des übergreifenden Testziels. Testschritte sollten nach folgenden Kriterien gewählt werden:

- Anbindung an *benannte Teilziele*, die durch den Test gewährleistet werden sollen;
- *Wechselseitige Unabhängigkeit*, um bei Auffinden eines Fehlers andere Teile der Funktionalität weiter erproben zu können;
- Umsetzbarkeit anhand von spezifischen *Testfällen*, definiert durch dokumentierte Testdatensätze geforderte Systemausgaben .

Die *Testorganisation* soll dabei Konflikte und Reibungsverluste bei Zusammenarbeit zwischen den einzelnen Mitgliedern der Testgruppe verringern und ihre zuverlässige Kommunikation mit den Entwicklern fördern. Unbedingt abzusprechen sind verbindliche Verfahren bei der Beseitigung von Fehlern, das heißt, es muß vereinbart werden, wer, wann und nach Rücksprache mit wem Änderungen in einer Komponente vornehmen darf. Dazu werden in der Regel Komponentenfassungen mit unterschiedlichem Zugriffsrecht gebildet.

Die *Testdaten* müssen unter Bezugnahme auf das definierende Dokument (Entwurf- bzw. Systemspezifikation) systematisch ausgewählt und einzelnen Testschritten zugeordnet werden. Da Tests niemals erschöpfend sein können, dient die systematische Auswahl von Testdaten einer möglichst weitgehenden Überdeckung der möglichen Eingaben. Dazu sind folgende Gesichtspunkte hilfreich:

- Die Bildung von *Fallklassen* in Abhängigkeit von gleichartigem erwarteten Ein-Ausgabe-Verhalten für alle Elemente der Klasse;
- Die Bildung von *Grenzwerten* zur Trennung zwischen erlaubten und nicht erlaubten Systemeingaben;
- Die Feststellung von *logischen Abhängigkeiten* bei möglichen Eingaben, um durch die Kombination und Reihenfolge der Testschritte ihre Aussagekraft zu erhöhen.

Zunächst sind alle Fallklassen bezogen auf erlaubte *Einzeleingaben* sowie ihre Abgrenzung von nicht-erlaubten Einzeleingaben abzuprüfen. Darauf aufbauend werden *Eingabekombinationen* erarbeitet, die individuell relevant und voneinander unabhängig sein und in ihrer Gesamtheit die Funktionalität des Systems möglichst weitgehend überdecken müssen. Testdaten werden in dokumentierte *Testfälle* zusammengefaßt und den einzelnen Testschritten zugeordnet.

Die *Testwerkzeuge* sind je nach der technischen Entwicklunsumgebung verfügbar und auszunutzen. Sie können die Erzeugung von Testdaten, von Testrahmen und von Testanweisungen an systematisch ausgewählten Programmstellen automatisieren oder die Auswertung von Testergebnissen ermöglichen oder erleichtern. Die Auslotung, rechtzeitige Erschließung und sinnvolle Anwendung von testwerkzeugen in der jeweiligen Entwicklungsumgebung ist eine wichtige Voraussetzung für den systematischen Test größerer Programmsysteme.

5.3 Produktverwaltung

Bei der arbeitsteiligen Entwicklung größerer Software entsteht ein Produkt als komplexe Struktur aus *Fassungen* von *Komponenten*. Verschiedene Fassungen einer Komponente sind entweder als zeitlich aufeinanderfolgende *Versionen* oder als zeitlich parallel existierende *Varianten* ausgeprägt.

Zum Verständnis dieser Struktur sind bereits bei der Herstellung im Erst-Zyklus drei Dimensionen zu berücksichtigen:

- der *Funktionsumfang* der Komponente bezogen auf die vereinbarte Systemversion oder eine vorgeschaltete Ausbaustufe,
- die *Einordnung* der Komponente in die beim Softwareentwurf als Grundlage für die Programmierung erarbeitete Systemzerlegung,
- der *Bearbeitungszustand* der Komponente bezogen auf den Grad ihrer Fertigstellung durch die Entwickler bzw. ihre Verfügbarkeit zum Testen und zur Systemintegration.

Bei der evolutionären Systementwicklung ist ferner das Zusammenspiel von Versionen bei der Herstellung und im Einsatz zu berücksichtigen. Bezogen auf jeden Einsatzkontext gibt es eine Folge von eingesetzten Systemversionen, die die Grundlage für die kundenspezifische Weiterentwicklung bilden. Bei mehrfach eingesetzten Systemen gibt es häufig zusätzlich Varianten für verschiedene Einsatzkontexte. Ein Produkt ist dann definiert als *Konfiguration* aus *Versionen* und *Varianten* von einzelnen Komponenten.

Die Produktverwaltung muß gewährleisten, daß der Umgang mit der Gesamtheit der unterschiedlichen Komponentenfassungen zuverlässig erfolgen kann, sodaß die jeweils

benötigten passenden Komponentenfassungen bei der Programmierung, der Validation und der Wartung ermittelt und bereitgestellt werden können. Dazu gehört die Erarbeitung eines *projektspezifischen Versionskonzepts*, in dem auch die geplanten Ausbaustufen berücksichtigt werden Die technische Grundlage dazu liefert eine rechnergestützte *Projektbibliothek*, deren Führung dem Produktverwalter untersteht.

Die Projektbibliothek muß die Möglichkeit bieten, die jeweiligen Fassungen der Komponenten nach den oben definierten Dimensionen zu charakterisieren, die Unterschiede zwischen verschiedenen Fassungen festzuhalten und den Übergang zwischen den Fassungen abzusichern.

Dazu werden die Fassungen mit *Attributen* versehen, die ihren Funktionsumfang, ihre Einordnung und ihren Bearbeitungszustand sowie die Zugehörigkeit zu einer Version oder Variante beschreiben. Die Vergabe und Änderung dieser Attribute erfolgt durch den Produktverwalter, wobei die Grundlagen dazu verbindlich festgelegt werden müssen. Dabei durchläuft eine Komponente eine Folge von definierten Zuständen, die in einem *Zustandsautomat mit definierten Übergängen* (z.B. "in Bearbeitung", "im Test", "freigegeben") beschrieben werden. Die Übergänge zwischen diesen Zuständen können dann mit Verfahren zur konstruktiven Qualitätssicherung (z.B. gemeinsame Verabschiedung von Dokumenten, Abnahme einer Komponente) verbunden werden.

Komponentenfassungen exisitieren häufig in mehreren Exemplaren, die bestimmten Bearbeitungszuständen entsprechen und mit speziellen *Zugriffsrechten* verbunden sind. Zu unterscheiden sind die Bearbeiterversion von der Testversion und der zur Integration freigegeben Version. Die Bearbeiterversion darf von den jeweiligen Entwicklern beliebig geändert werden. Die Testversion ist nur für die Testgruppe zugänglich und dient zur Kommunikation über Fehler. Die freigegebene Version darf von niemandem geändert aber von allen zur Systemintegration benutzt werden.

Die zuverlässige Verwaltung dieser Fülle von Komponentenfassungen ist eine unerläßliche Voraussetzung für das Gelingen der Zusammenarbeit bei der Softwarerealisierung und Weiterentwicklung. Sie muß von vornherein eingeplant und auf einsichtige Konzepte gegründet sein.

6 Projektbegleitende Dokumentation

Softwareentwicklung ist mit der Erstellung von Dokumenten untrennbar verbunden. Das Produkt Software ist selbst durch Dokumente definiert: Programme, die durch Kommentare zu erläutern sind sowie die Grundlagen für ihre Herstellung und alle Zwischenprodukte im Phasenmodellsind Texte. Unter Dokumentation wird darüberhinaus die Herstellung von ergänzenden Texten für verschiedene Zwecke verstanden:

- die *Produktdokumentation* besteht aus allen erläuternden Texten zum Produkt; dabei sind zu unterscheiden die *technische* Dokumentation als Grundlage für die Pflege und Weiterentwicklung des Systems und die *Benutzerdokumentation*, die das System für den Gebrauch erschließt; die Produktdokumentation muß sich auf die jeweilige Systemversion beziehen und mit dem Produkt zusammen weiterentwickelt werden;

- die *Prozeßdokumentation* dient zum Festhalten der Historie des Entwicklungsprozesses; sie legt die Planung der Referenzlinien einschließlich Zuständigkeiten und Qualitätskriterien fest, sie ermöglicht die Rekonstruktion von getroffenen Entscheidungen über das Produkt und die Koordination der Zusammenarbeit, sie enthält sämtliche Unterlagen zur Basismaschine, die Testdaten und Testprotokolle sowie die verabschiedeten Konventionen.

Die technische Dokumentation besteht aus den kommentierten Programmen, den definierenden Dokumenten, den Zwischenprodukten (Meilensteinen und Referenzlinien), den Validationsunterlagen und dem in der Projektbibliothek verankerten Versionskonzept.

Konventionen über die *Kommentierung* von Programmen müssen vereinbart und von der Qualitätssicherung überprüft werden. Sie sollten die Funktionsweise jedes Programmteils (Modul, Prozedur, Typ) und seine Wechselwirkung mit anderen erläutern. Bei der Kommentierung von Algorithmen sollten die Wahl der Programmanweisungen begründet werden, es geht nicht darum, den Code textuell zu duplizieren.

Die *definierenden Dokumente* sind die Anforderungsdefinition, die Systemspezifikation und die Entwurfsspezifikation. Ihre Inhalte sind durch das Phasenmodell vorgeschrieben, müssen aber konkret für das Projekt vereinbart werden. Sie bestehen aus formalisierten Anteilen je nach der gewählten Spezifikationsmethode und aus erläuternder Prosa. Zu beachten ist der jeweilige Adressatenkreis: während die Anforderungsdefinition und die Systemspezifikation für die Kommunikation zwischen Entwicklern und Benutzern gemeint ist, wendet sich die Entwurfsspezifikation ausschließlich an die Entwickler.

Zwischenprodukte können neben definierenden Dokumenten zum Beispiel auch Begriffsglossare, Modelle oder Prototypen sein.

Das *Benutzerhandbuch* muß das System für den Gebrauch erschließen. Zum Unterschied von der technischen Dokumentation muß es daher in der Fachsprache der Benutzer geschrieben werden und das System als Arbeitsmittel bzw. Kommunikationsmedium in den Arbeitsprozessen erläutern. Dabei sind sowohl einfährende Texte als auch Übersichten zum Nachschlagen vorzusehen.

Die *Validationsunterlagen* beinhalten die vereinbarte Teststrategie und die Testorganisation, die kommentierten Testadtensätze und die Testprotokolle. Im Falle von Optimierung kommen noch die Grundlagen und Ergebnisse von Laufzeitmessungen dazu.

Die Prozeßdokumentation muß sowohl individuell als auch für das Team als Ganzes geführt werden. Dabei sind die situativ gegebenen Kommunikationsstrukturen zu berücksichtigen. Ein allgemein zugänglicher *Projektordner*.ist eine wesentliche Hilfe zur Herausbildung eines gemeinsamen externen Gedächtnisses. Er sollte alle getroffenen Entscheidungen, die Besprechungsprotokolle, die Referenzlinien, die vereinbarten Arbeitskonventionen und aufgetretene Fehler und Hindernisse enthalten und laufend auf den neuesten Stand gebracht werden. Sein Zustand muß vom Projektleiter oder von der Qualitätssicherung sorgfältig im Auge behalten werden.

Ein *Projekttagebuch*, in dem alle projektrelevanten Ereignisse mit Datum und Namen der betroffenen Personen festgehalten werden, ist ein wesentliches Hilfsmittel, um den Entwurfsprozeß gemeinsam zu reflektieren.

Für die laufende Kommunikation sollten neben den Besprechungen und persönlichen Rücksprachen sowohl Anschlagtafeln und ähnliche physische kommunikationsunterstützende Mittel als auch die elektronische Kommunikation über den Rechner in transparenter Weise genutzt werden.

Je nach den situativen Gegebenheiten wird die Dokumentation ganz oder teilweise am Rechner geführt.

7 Gestaltbildende Projekttechniken

Für eine erfolgreiche Softwareentwicklung kommt es darauf an, daß die Gemeinschaft von Entwicklern und Benutzern anhand des vorgegebenen Entwicklungsauftrags "Geist" entfaltet, indem sie eine gemeinsames Verständnis über das zu entwickelnde Produkt, den Entwicklungsprozeß und die Einbettung des Produktes in seinen Einsatzkontext herstellt und aufrechterhält. Dies kann durch eine sorgfältige Projektgestaltung gefördert werden.

7.1 Perspektiven einnehmen und kreuzen

Eine *Perspektive* ist eine Klasse zusammengehöriger Sichten auf ausgewählte Aspekte eines Gegenstandsbereichs aus einem bestimmten Blickwinkel. Zu unterscheiden sind *implizit vorhandene* Perspektiven, die in der Kommunikation herausgearbeitet und anerkannt werden müssen und *explizit ausformulierte* Perspektiven, die bewußt eingenommen werden können. Das bewußte Einnehmen und Kreuzen von Perspektiven eröffnet Chancen zu einer sinnvollen und reichhaltigen Zusammenarbeit. Ein multiperspektivischer Ansatz bei der kooperativen Systementwicklung kann folgende Perspektiven deutlch machen und verbinden:

- den *persönlichen Horizont* aller Beteiligten, der sämtliche Erkenntnisse subjektiv färbt und in der individuellen Entwicklung begründet ist;
- *rollenspezifische Sichten*, die sich aus unterschiedlichen Aufgaben bei der Herstellung und beim Einsatz ergeben; z. B. die Sicht des Projektleiters, der Entwickler, der Testgruppe, von Managern oder die von Benutzern mit verschiedenen Aufgabenbereichen;
- *Positionen*, die auf Interessen und Werten beruhen, die Individuen oder Gruppen vertreten; z. B. Positionen des Managements oder gewerkschaftlicher Interessenvertreter;
- *Leitmetaphern*; die das Verständnis der Softwareherstellung oder der Softwarebenutzung sowie die Interpretation der hierbei eintretenden spezifischen Situationen stützen; z.B. die Werkzeug-, die Schreibtisch- oder die Kommunikationsmetapher für den Computer;
- vereinbarte *Gestaltungsanliegen*; die mit Zielen und Qualitätskriterien verbunden sind und für die Orientierung und Prioritätensetzung in Entscheidungsituationen den Ausschlag geben.

Eine bewußte multiperspektivische Herangehensweise strebt an, durch das Einbeziehen der Unterschiedlichkeit, vertiefte Einsichten über Wege zu finden, die für alle gangbar sind. Ferner soll der gemeinsame Erkenntnis- und Entscheidungsprozeß für alle Beteiligten transparent werden.

7.2 Grundlagen der Zusammenarbeit legen und erneuern

Hier geht es um die Erarbeitung von gemeinsam getragenen Organisationsformen und Konventionen. Diese können nicht nur von außen vorgegeben oder aufgezwungen werden, sondern müssen im Entwicklungsprozeß erarbeitet, bezogen auf das konkrete Projekt, diskutiert und angepaßt werden. Wichtige Einzelaspekte sind:

- Welche Gültigkeit haben Rahmenbedingungen und Festlegungen in definierenden Dokumenten? Was kann als fest angenommen werden und wo sind langfristig Änderungen zu erwarten?
- Nach welchen Modalitäten wird die Terminplanung durchgeführt bzw. revidiert?
- Warum werden wann Prototypen bzw. Vorversionen erstellt und wie werden diese ausgewertet?
- Für welche Aufgaben werden übergeordnete Gruppen gebildet und wie arbeiten diese mit den übrigen Projektmitgliedern zusammmen? Wie werden prozeßbezogene Aufgaben mit produktbezogenen Aufgaben koordiniert?
- Nach welchen Konventionen wird gearbeitet? Auf welche Grundlagen bezieht man sich bei Entwurf, bei der Schnittstellenfestlegung, bei der Programmierung?
- Wie werden Testdaten und Teststrategien entworfen?
- Wie wird Versionsverwaltung initialisiert und fortgeschrieben?

Dies sind herkömmliche und in allen Projekten anfallende Aufgaben. Sie werden dann methodisch umgesetzt, wenn sie von vornherein bewußt anerkannt und mit dem Projekt zusammen weiterentwickelt werden.

7.3 Eine Projektsprache herausbilden

Eine wichtige Voraussetzung für die kooperative Systementwicklung ist die wechselseitige Verständigung über den interessierenden Realitätsbereich. Entwickler und Benutzer müssen zum einen die relevanten Begriffe aus der jeweiligen Fachsprache verbindlich rekonstruieren und hinreichend eindeutig zur späteren Formalisierung festlegen. Zum ande-

ren gilt es, die technische Welt des Rechners in einer für die Benutzer verständlichen Form zu erschließen, wobei auch Metaphern über den Rechner und seinen Einsatz hilfreich sein können. Begriffserklärungen und Festhalten der vereinheitlichten Bedeutung zunächst im Informellen, aber schrittweise auch formalisiert und mit den jeweils vorhandenen Werkzeugen unterstützt, festigen den gemeisamen Sprachgebrauch. Da das Verständnis der zu leistenden Arbeit sich im Verlauf des Projekts entfaltet und ändert, muß seine historische Entwicklung erschließbar sein. Daher sind Besprechungsergebnisse nach Schwerpunkten geordnet und mit Begründung im Projektordner zugänglich zu machen.

7.4 Rollen definieren, wahrnehmen und austauschen

Verantwortung für Aufgaben wird durch Personen wahrgenommen. Das Konzept der funktionellen Rollen dient zur Beschreibung dieses Zusammenhanges. Die *Verantworlichkeit* für eine funktionelle Rolle kann während eines Projektes zwischen den Mitgliedern *wechseln*. Die Rollen können dazu benutzt werden, unterschiedliche Perspektiven auf ein Projekt herzustellen.

Zur *Einbeziehung der Benutzerperspektiven* in die Softwareentwicklung kann auch die Etablierung eines sogenannten *virtuellen Benutzer*s dienen, wenn die realen Benutzer nicht bekannt sind oder nicht zur Verfügung stehen oder wenn sie als Ergänzung zur realen Kommunikation im Entwicklungsteam ihre Vertrauensperson haben sollen. Das bedeutet konkret, daß in Besprechungen und bei der Verabschiedung von Dokumenten ein Projektmitglied die Rolle der prospektiven Benutzer übernimmt und die Tauglichkeit der vorgeschlagenen Lösung im Hinblick auf seine (vorgestellten) Arbeitsaufgaben untersucht. Bei komplexeren Systemen mit unterschiedlichen Benutzergruppen müßte man mehrere virtuelle Benutzer mit unterschiedlichen Rollen vorsehen.

Dieses Konzept ist eine spezielle Ausprägung von *Rollenaustausch und -verschmelzung*, durch die die Einseitigkeit des eigenen Blickwinkels (z.B. als Entwickler) überwunden werden kann. Eine Rollenverschmelzung ergibt sich als wesentlicher Aspekt im Zusammenhang mit Prototyping, wenn sich Entwickler der Konsequenzen ihrer Entwurfsentscheidungen aussetzen, indem sie entweder selbst das System benutzen, oder als Zuhörer an Probebenutzungen anderer teilnehmen.

7.5 Einbezogenheit und Vernetzung realisieren

Die produktbezogene Arbeitsteilung bei der Softwareentwicklung ist eine wesentliche Voraussetzung für die Herstellung größerer Software. Gleichzeitig muß aber ein ganzheitliches, gemeinsames Verständnis über die Einordnung dieser Teilprodukte in ihren

Kontext zustandekommen und erhalten bleiben. Dazu bieten sich folgende Maßnahmen an:

- Austausch von Arbeitsdokumenten z.B. beim Übergang zwischen Phasen oder von einer Ausbaustufe zur nächsten,
- gemeinsames Überprüfen von Arbeitsdokumenten anhand von Kriterien, ausgewählten Testdaten und den Anforderungen unterschiedlicher Rollen,
- Bildung von übergeordneten Gruppen für prozeßbezogene Aufgaben und Streuung des so vorhandenen zentralen Wissens durch Wiedereingliedern von Verantwortlichen in die produktbezogene Arbeit,
- Rotieren von übergeordneten Aufgaben zwischen den einzelnen Mitgliedern.

7.6 Konstruktive Kritik geben und annehmen

Bei der Softwareentwicklung geht es darum, aus den immer nur vorläufigen Teilergebnissen einzelner Projektmitglieder ein für alle gemeinsames Ergebnis zu erarbeiten. Dabei ist es wesentlich, Zwischenprodukte gemeinsam zu bewerten und einzuordnen. Zur Bewertung von Dokumenten sowie von Prototypen ist *konstruktive Kritik*, etwa in Form von Design-Reviews oder eines Autor-Kritiker-Zyklus, von hervorragender Bedeutung. Konstruktive Kritik muß versachlicht sein und auf vereinbarten Grundlagen beruhen. Diese können zum Beispiel sein: Projektspezifische Qualitätsanforderungen, festgelegte Kritikschwerpunkte mit darauf abgestimmten Fragestellungen, typische Arbeitsprozesse zur Untersuchung der Aufgabenangemessenheit, systematisch ausgewählte Testdaten zur Validation. Ganz wesentlich erweist sich dabei die Art, wie Kritik geäußert wird und die Bereitschaft, Kritik anzunehmen. Wir halten es für notwendig das Geben und Annehmen von konstruktiver Kritik in Projekten sorgfältig vorzubereiten, einzuüben und zu betreuuen. Konstruktive Kritik ist nur dann umsetzbar, wenn die Beteiligten in der Lage sind, den Wert der Arbeit des jeweils andern anzuerkennen und zu würdigen.

7.7 Milieu zur Zusammenarbeit bilden und lebendig erhalten

Dies ist nur in Gemeinschaften denkbar, in denen Menschen sich wechselseitig annehmen und vertrauen können. Dies setzt die Bereitschaft voraus, über Interessenkonflikte hinweg die individuellen Gegebenheiten aller Beteiligten und ihre Entfaltungsmöglichkeiten zu berücksichtigen. Wechselseitiges Vertrauen kann nicht durch ein manipulatives Vorgehen erreicht, aber sein Entstehen kann durch Sorgfalt und Aufeinander-Eingehen gefördert werden.

Voraussetzung ist die Gelegenheit, sich besser kennenzulernen, um einen reichhaltigen persönlichen Erfahrungshintergrund zu bilden. So entstehen wichtige informelle Kommunikationskanäle, die für die Gestaltung von fruchtbaren Besprechungen, für die zu Einsichten führende gegenseitige Würdigung und Kritik sowie für die verantwortungsvolle Identifikation mit der gemeinsamen Aufgabe unerläßlich sind. Die Aufrechterhaltung einer vertrauensvollen Gemeinschaft ist eine wesentliche Aufgabe des Projektleiters, der dazu die persönliche Befähigung aber auch einen ausreichenden Entscheidungsspielraum braucht.

Die von den Beteiligten erlebte Eingebundenheit in der Gemeinschaft ist von ausschlaggebender Bedeutung für den Projekterfolg. Ein Entwurf ist vor allem dann gut, wenn das Team sich mit ihm identifiziert und ihn verinnerlicht. Konventionen haben nur dann Wert, wenn sie akzeptiert und umgesetzt werden. Qualität kann nur dort entstehen, wo dies den Beteiligten ein wichtiges Anliegen ist.

Index

ablauforientierten Modellierung 6
Abnahmetest 26
Abstrakte Datentypen 9
adaptionsfähigen Basissystems 21
Akzeptanzfähigkeit 6
Anforderungen 4; 5; 6; 8; 9; 12; 13; 15; 18; 20; 25; 36
Arbeitserleichterung 6
arbeitsteilige Programmierung 9
arbeitsteiligen Programmierung 8
Attributen 30
Aufgabenangemessenheit 7
Ausfallsicherheit 7
äußere Architektur 7
Auswertung 12; 17; 19; 20; 29
Automatisierungsfähig 5
Automatisierungswürdig 5
Autor-Kritiker-Zyklus 25; 36
Bearbeitungszustand der Komponente 29
benannte Teilziele 28
Benutzerdokumentation 31
Benutzerhandbuch 3; 32
Benutzt-Hierarchie 9
Benutzungsschnittstelle 7
Benutzungsschnittstelle 7
Betriebsorientierte Qualitätsmerkmale 7
Bewältigung der Komplexität 8
Bottom-Up-Test 27; 28
Datenintegrität. 7
definierende Dokumente 12
definierenden Dokumente 31
Dokumentation 3; 11; 12; 31; 32
Effektivität 5
Einbeziehung der Benutzerperspektiven 35
Eingabekombinationen 29
Einordnung der Komponente 29
Einsatz 3
einsatzspezifischen Varianten 13
Einzeleingaben 26; 29
Endliche Automaten, 9
Entscheidungstabellen 9
Entwickler 24
Entwicklung einer neuen Version 13
Entwicklungsschritte 2; 3; 4; 15; 20
Entwicklungsstrategien 3
Entwicklungsumgebung 11; 12; 28; 29
Entwurfsspezifikation 9; 10; 11; 21; 26; 31
Ergebnisse 2
Erstzyklus 18
Erwartungskonformität 7
Etablierung 3; 17; 25; 35
Evaluation 12; 13
evolutionäres 3
evolutionäres Prototyping 20
experimentelles 3
experimentelles Prototyping 15; 16
explizit ausformulierte Perspektive 33
exploratives 3
exploratives Prototyping 15
Fallklassen 28; 29
Fassungen von Komponenten 29
Fehlerlokalisierung 27
Fehlertoleranz 7
Flexibilität 6; 27
Folge von Systemversionen 15
funktionelle Rollen 24
funktionelle, Leistungs-, Handhabungs- und Einbettungsanforderungen 6

funktionellen Rollen 5
Funktionsauswahl 20
Funktionsumfang 19; 29; 30
Gestaltung der Benutzungsschnittstelle 21
Gestaltungsanliegen 33
Grenzwerten 28
Herstellung 2; 3; 8; 14; 15; 18; 20; 24; 29; 31; 33; 35
horizontales Prototyping 20
implizit vorhandene Perspektive 33
innere Architektur 8
Integrations- und Systemtest 26; 27
intellektuelle Beherrschbarkeit 9
Interviewtechniken 6
Kommentierung von Programmen 31
Komponententest 10; 26; 27
Konfiguration 13; 29
Konstruktion 20
konstruktive Kritik 36
Konstruktive Qualitätssicherung 3
Konstruktiver Kritik 25
Konventionen 25
Konventionen zur Kommentierung von Programmen 26
Konventionen zur Nutzung 26
Konzept zur Versions- und Produktverwaltung 10
Koordination des Entwicklungsprozesses 2; 19; 22
Koordination des Softwareentwicklungsprozesses 17
Korrektheit 10; 11; 12; 26
Kriterien zur Bewertung 22
Laufzeiteffizienz 11
Layout-Konventionen 25
Leitmetaphern 33
Lesbarkeit 11; 25
Liste von Zwischenprodukte 22
logische Abhängigkeiten 4
logischen Abhängigkeiten 28
Lokalität 9
Meilensteine 3; 22; 31
Meilensteinen 22
Minimalität der Schnittstellen 9
Modell der Objekte und Operationen 7
Moduln 9; 27
Namenskonventionen 26
Nutzung 17
Ökonomie des Steuer- und Datenflusses 9
Optimierung 11; 12; 17; 32
persönlichen Horizont 33
Perspektive 24; 33; 35
Pflege 13; 15; 17; 31
Pflege der aktuellen Version 13
Phasenmodell 2; 4; 15; 16; 17; 26; 31
Platzhalter 27; 28
Positionen 33
problemorientierte Änderbarkeit 7
produktorientierte Sichtweise 2
Produktverwaltung 3
produkt- 2
Produktdokumentation 3; 31
Produktverwaltung 11; 29; 30
Programmierkonventionen 25
Programmskeletts 21
Projekt 2; 3; 4; 6; 17; 19; 21; 24; 27; 31; 34; 35; 36
Projektassistent 24
Projektbegleitende Dokumentation 31
Projektbibliothek 10; 30; 31
Projektetablierung 17; 18; 22
Projektgestaltung 3; 24; 33

Projektleiter 24; 32; 33; 37
Projektordner 32; 35
Projektplan 17; 19; 22
Projektplans 3
projektspezifischen Versionskonzepts 30
projektspezifisches 2
Projekttagebuch 32
Projekttechniken 2
Prototyping 3; 15; 19; 20; 21
prozeduraler Schnittstellen 9
prozeßbezogene Aufgaben 3; 36
prozeßbezogenen Aufgaben 2
Prozeßdokumentation 3; 31; 32
prozeßorientierte Sichtweise 2
Pseudocode 9
Referenzglossare 6
Referenzlinien 3; 22; 23; 25; 31; 32
Relevanz 5
Review 22; 25; 36
Revisionsetablierung 3
Revisionsetablierung 17; 18; 22
Revisionszyklus 18
Robustheit 7
Rollenaustausch und -verschmelzung 35
rollenspezifische Sichten 33
Schnittstellen 9; 11; 24; 26; 34
Selbstbeschreibungsfähigkeit 7
situativ adaptiert 3
Softwarerealisierung 15; 16; 30
Sozialverträglichkeit 5
Speicherökonomie 11
Steuerbarkeit, 7
Struktogramme 9
Strukturelle Änderbarkeit 9
stufenweise Abstraktion 9
systematische Auswahl von Testdaten 11; 28
Systematischer Test 3; 27
Systemgestaltung 6; 7; 15; 16; 20; 21; 24
Systemgestaltung, 15
Systemkern 27
Systemkonzept 5; 17
Systemspezifikation 8; 11; 15; 17; 21; 22; 26; 28; 31
Systemversion 13; 14; 17; 29
Systemversionen 3
Tabellen 9
technische Dokumentation 31
technische Änderbarkeit 7
Terminplanung 27; 34
Testdaten 11; 27; 28; 29; 31; 34; 36
Testfälle 26; 29
Testfällen 28
Testorganisation 11
Testorganisation 28; 32
Testrahmen 11; 27; 28; 29
Testschritte 11; 28; 29
Teststrategie 11; 27; 32; 34
Testwerkzeuge 11; 27; 28; 29
Top-Down-Test 27
Überprüfen von Dokumenten 25
Umfeldvorbereitung 17
Umweltverträglichkeit 5
Validationsunterlagen 11; 32
Varianten 13; 29
Verantworlichkeit 35
Verfahren zur Bewertung 22
Verifikation 10
Versionen 13; 15; 18; 29
Verständlichkeit 9
vertikales Prototyping 20
Virtueller Benutzer 24; 35
vollständige oder teilweise Simulation 21
Wartung 13; 15; 30

Wechselseitige Unabhängigkeit 28
Weiterentwicklung 3; 12; 13; 15; 31
Weiterentwicklungsorientierte Qualitätsmerkmale 7
Weiterverwendung 20
Wiederverwendbarkeit 9
zeitlich aufeinanderfolgenden Versionen 13
zeitlichen Abwicklung 2
zeitlichen Projektabwicklung 14
Zugriffsrechten 30
Zusatzaufwand 28
Zustandsautomat mit definierten Übergängen 30
Zwischenprodukte 3; 22; 25; 31; 32; 36
Zyklen 3; 15; 16; 22
zyklische Modell 2