2 Software Engineering Projekte als Grundlage für Qualitätsmanagement

Da Software Engineering eine sehr junge Ingenieursdisziplin ist, gibt es im Gegensatz zu anderen Disziplinen – bei denen bereits seit Jahrhunderten professionell gearbeitet und geforscht wird – noch unzureichende professionelle Methoden und Techniken.

Am Anfang der Entwicklung steht das Handwerk mit der Herstellung von Waren durch besonders talentierte Amateure. Um für den dadurch entstandenen Markt konkurrenzfähige Produkte zu etablieren, wird die Produktion systematisiert und rationalisiert. Hier werden bereits geübte Arbeiter eingesetzt. Die parallel dazu verlaufende Forschung, die zur Innovation beitragen will, ergibt gemeinsam mit der Wirtschaft den Kern einer Ingenieursdisziplin, welche durch die Analyse von Strukturen und Fakten gekennzeichnet ist. Die Weiterentwicklung durch Profis führt schlussendlich zu einer Segmentierung des Marktes.

Diese Entwicklung können wir auch beim Software Engineering beobachten: Anfang der 80er Jahre verstand man unter Software-Entwicklung (SE) höchstens das Programmieren von Recheneinheiten – die Wissenschaft beschränkte sich auf die Erforschung von Algorithmen und Datenstrukturen. 1968/69 wurde erstmals von der NATO gefordert, Software Engineering als Ingenieurdisziplin anzuerkennen und sie nicht nur als Kunst zu sehen. Als in den 60er und 70er Jahren der Begriff der „Software-Krise“ (sehr schlechte Qualität der erzeugten Systeme) geformt wurde, wurden erstmals praxiserprobte Vorgehensmodelle veröffentlicht.

Als in den 80ern formalisierte Entwicklungsmethoden endlich einer breiten Masse zugänglich gemacht wurden, glaubte man schon an den Übergang von Wissenschaft und Wirtschaft zu einer professionellen Ingenieursdisziplin. Leider ist dieser Wechsel noch immer nicht vollständig vollzogen, weshalb die Weiterentwicklung von Methoden und Vorgehensmodellen für die Projektarbeit noch immer das primäre Ziel ist.

Die folgenden Abschnitte beschreiben Typen von Softwareprodukten und -projekten, Projektmanagement-Aktivitäten sowie Personen und Rollen wie sie in einem Projekt vorkommen. Der letzte Abschnitt gibt einen Überblick über die wesentlichen Schritte eines Softwareentwicklungsprozesses.

2.1 Typisierung von Softwareprodukten und -projekten

Software durchdringt ein weites Spektrum an Anwendungen und Domänen; zudem ist Software und die zugehörige Computertechnologie einem rasanten Wandel unterworfen. Jeder Versuch, diese Vielfalt eindeutig zu typisieren, muss fehlschlagen – trotzdem lassen sich Bereiche und Typen von Projekten grob herauskristallisieren, die seit mehreren Jahren industrierelevant sind.

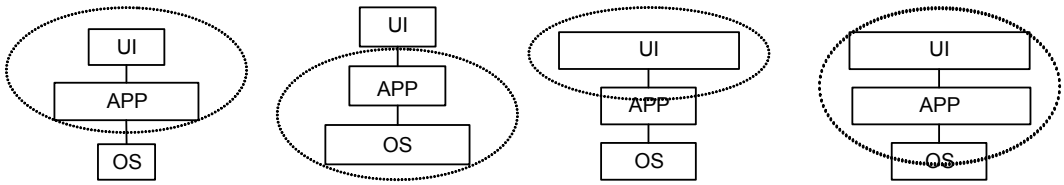
Die folgenden beiden Abschnitte beschreiben Charakteristika von verschiedenen Softwaresystemen und Projekttypen.

2.1.1 Systemtypen

Software begleitet uns in allen möglichen Bereichen des Lebens. Sie unterscheidet sich nach Art und Typus. Wir können die meisten Software-Systeme einem der folgenden Typen zuordnen:

- Kommerzielle Applikationen
- Echtzeitsoftware
- Web-Applikationen
- Computerspiele

Dies sind zwar die am häufigsten vertretenen Typen, allerdings existieren, wie oben bereits erwähnt, auch andere (Expertensysteme, Systemsoftware etc.). System-Typen unterscheiden sich in der Produktstruktur, im Entwicklungsprozess und im Qualitätsfokus der Endbenutzer. Die Produktstruktur eines Systems beinhaltet drei verschiedene Schichten an Funktionalität:



a) Kommerzielles System b) Echtzeit c) Webapplikation d) Computerspiele

UI ... User Interface (Benutzerschnittstelle)
APP … Application (Anwendung)
OS … Operating System (Betriebssystem)

Abbildung 2.1: Systemtypen im Überblick

Benutzerschnittstelle (UI): Die Benutzerschnittstelle oder das User Interface (UI) ist diejenige Komponente des Software-Systems, die Interaktionen zwischen Mensch und Maschine behandelt. Dies involviert Software und Hardware für Ein- und Ausgabe. Eine Benutzerschnittstelle kann z.B. aus Tastatur, Maus, Bildschirm und Drucker als Hardware-Komponenten, und graphischen oder alphanumerischen Elementen, Bildern und Anweisungen in Kommandozeilen als Software-Komponenten bestehen.

Applikations-Logik (APP): Die Applikations-Logik ist die Summe aller Funktionen, die speziell für die Applikation geschrieben wurden, und die vom Benutzer nicht „gesehen“ werden. Sie werden entweder durch Aktionen an der Benutzerschnittstelle, durch Bedingungen in anderen Komponenten des Systems (z. B. Sensoren) oder durch zeitliche Bedingungen ausgelöst. Typische Funktionen der Applikations-Logik sind Berechnungsfunktionen (z.B. Berechnung einer Versicherungsprämie), Datenbankfunktionen oder Funktionen zur Aktorenkontrolle.

Betriebssystem (OS): Die unterste Stufe in einem Software-System umfasst Aufrufe von Systemfunktionen, z.B. Synchronisations-Mechanismen wie Semaphore, Timing-Funktionen oder Lese- und Schreibkommandos an I/O-Ports.

Die Struktur eines Systems beeinflusst die relative Bedeutung von Qualitätskriterien. Ein Qualitätskriterium kann in einem System-Typ wichtiger sein als in einem anderen, was sich auf die Maßnahmen des Qualitätsmanagements auswirkt.

Kommerzielle Applikationen

Bei kommerziellen Applikationen (siehe Abbildung 2.1a) liegt der Schwerpunkt auf der Applikationslogik. Kommerzielle Applikationen schließen auch Browser-gestützte E-Commerce-Applikationen oder andere Web-Applikationen mit Datenbank mit ein.

Die wichtigste Komponente dieses Systems ist diejenige, die Daten aus der Datenbank holt, verarbeitet und speichert. Die Benutzerschnittstelle ist eher zweitrangig, und Aufrufe von Systemfunktionen werden meist indirekt über von der Programmiersprache bereitgestellte Funktionen aufgerufen.

Folglich werden die meisten kommerziellen Applikationen mit High-Level-Programmiersprachen entwickelt. Diese bieten standardisierte Objekte als Komponenten der Benutzerschnittstelle. Daher umfassen kommerzielle Applikationen eine hohe Bandbreite an Funktionalität mit vergleichsweise geringen Variationen in den Komponenten der Benutzerschnittstelle.

Das wichtigste Qualitätskriterium für kommerzielle Applikationen ist Funktionalität, gefolgt von Zuverlässigkeit und Effizienz. Benutzbarkeit, Wartbarkeit und Portabilität sind – in den meisten Fällen – zweitrangig.

Echtzeitsysteme

Echtzeitsysteme (siehe Abbildung 2.1b) hängen sehr stark von ihrem Zeitverhalten ab. Sie bestehen hauptsächlich aus zeitkritischen Funktionen, Low-Level-Funktionen nah am Betriebssystem-Kernel und Aufrufen von Funktionen des Betriebssystems selbst. Die Benutzerschnittstelle ist auf die notwendigsten Funktionen für Ein- und Ausgabe gestutzt. Die Applikationslogik ist eher eingeschränkt, da zu viel Funktionalität das Zeitverhalten des Systems negativ beeinflusst.

Das wichtigste Qualitätskriterium für Echtzeitsysteme ist Effizienz, gefolgt von Zuverlässigkeit, Funktionalität und Wartbarkeit. Benutzbarkeit und Portabilität sind eher zweitrangig.

Web-Applikationen

Web-Applikationen (siehe Abbildung 2.1c), deren einzige Aufgabe es ist, eine Firma oder eine Person zu repräsentieren, d.h. Homepages, haben ihren Schwerpunkt auf einem attraktiven Layout. Nicht eingeschlossen dabei sind E-Commerce-Applikationen und andere Web-Applikationen, die eine Datenbank benutzen.

Die Benutzerschnittstelle ist überaus wichtig und die im Entwurf teuerste Komponente. Web-Applikationen benutzen vordefinierte Browser oder Web-Server und bieten wenig bis gar keine Funktionalität, sodass Applikationslogik und Gebrauch von Funktionen des Betriebssystems minimal sind.

Das wichtigste Qualitätskriterium für Web-Applikationen ist Benutzbarkeit, gefolgt von Leistung und Zuverlässigkeit – man denke an Hochlast beim Zugriff auf Webseiten. Die anderen Kriterien sind eher vernachlässigbar.

Computer-Spiele

Benutzer von Computer-Spielen (siehe Abbildung 2.1d) verlangen der Benutzerschnittstelle des Spiels sehr viel ab, und sie kaufen eher Spiele, die neue und umfangreiche Funktionalität mit einer vernünftigen Effizienz bieten. Daher wird viel Aufwand in die Benutzerschnittstelle und die Applikationslogik investiert. Teilweise werden Funktionen des Betriebssystems direkt aufgerufen oder sogar durch selbst entwickelte Funktionen ersetzt, um Performance zu gewinnen. Viele Hersteller von Computer-Spielen benutzen keine High-Level-Programmiersprachen, sondern verwalten ihre eigenen Objekt- und Funktions-Bibliotheken. Diese Faktoren führen zu nicht-standardisierten Benutzerschnittstellen und einer großen Anzahl von Funktionen.

Die wichtigsten Qualitätskriterien sind Benutzbarkeit, Funktionalität und Effizienz. Zuverlässigkeit, Wartbarkeit und Portierbarkeit sind eher unwichtig.

2.1.2 Projekttypen

Ein Software-Projekt ist erfolgreich, wenn es zeitgerecht abgeschlossen wird, den Kundenanforderungen entspricht und die Produktionskosten nicht zu sehr den vorgegebenen Rahmen überschreiten.

Bei der Planung des Projektes ist daher auf die im folgenden Abschnitt beschriebenen Größenordnungen zu achten.

Charakteristika eines Projekts

- *Komplexität, Funktionalität*: Die Komplexität eines Software-Produktes steigt überproportional mit seiner Größe, da die Anzahl der Beziehungen zwischen seinen Teilen exponentiell mit deren Anzahl steigt. Diese Komplexität ist Ursache für viele klassische Softwareprobleme. Folgen sind oft mangelnde Kommunikation zwischen den Teammitgliedern, erschwerte Planung, Übersicht und Kontrolle des Projektes. Ebenso verhält es sich mit wachsenden Ansprüchen an die Funktionalität des Produktes.
- *Qualitätsansprüche*: Qualität ist die Erfüllung gegebener Erfordernisse. Grundlegende Qualitätsmerkmale sollten zu Projektbeginn von und mit dem Kunden festgeschrieben werden. Wird dies nicht getan, so werden diese meist von einem Programmierer beim Codieren ohne Wissen über größere Zusammenhänge spontan festgelegt. Dies führt regelmäßig zu unerwünschten Folgen und zusätzlichem Arbeitsaufwand.
- *Anzahl der am Projekt beteiligten Personen*: Die Anzahl der beteiligten Entwickler, die entscheidenden Einfluss auf den Verlauf des Projektes haben, ist eine wesentliche Kennzahl für die Größe des Projektes. Je größer die Anzahl der Entwickler und damit die Größe des Projektes, desto wichtiger ist die effiziente Kommunikation mit entsprechender Projektdokumentation. Es ist keine Übertreibung, dass ab einer Mitarbeiteranzahl von fünf Personen die Kommunikation von Projektinformation wohl das entscheidende Kriterium für

das Gelingen des Projektes ist. (Sogar wichtiger als die technische Kompetenz der Beteiligten, sofern ein Mindestmaß an technischen Kenntnissen verfügbar ist.)

- *Zu erwartende Einsatzdauer des Softwareproduktes*: Im Bereich von z.B. Großrechneranwendungen verbleiben Teile der Software bis zu 20 Jahre im Einsatz. Vollständige und qualitativ hochwertige Dokumentation der Entwicklung und der laufenden Modifikationen der Software ist unerlässlich, da die Software spätestens nach 10 Jahren personenunabhängig wartbar sein muss.

2.1.3 Projektgrößen

Ein weiteres wesentliches Unterscheidungskriterium wird durch die Größe des Softwareprojektes bestimmt. Sowohl die Auswahl der Software-Engineering Methoden als auch die Anwendung von qualitätssichernden Methoden wird sehr stark von diesem Einflussfaktor bestimmt. Bei "Ein-Personen-Projekten" mit kurzer Projektlaufzeit wird der Entwickler weitgehend auf definierte Vorgehensmodelle und spezifische Werkzeuge der Qualitätssicherung verzichten, während deren Anwendung bei großen und komplexen Projekten unumgänglich ist.

Tabelle 2.1 zeigt verschiedene Kategorien der Größenordnungen von Software-Engineering Projekten. Die angegebenen Kategorien verstehen sich selbstverständlich nicht als Fachtermini.

Größe	Beschreibung	Aufwand	zusätzliche Dokumentation	Beispiel
super light (SL)	einer für sich selbst	0.1	keine	Rechenprobleme, einfache Spiele, Algorithmen
light (L)	einer für einen Kunden/Anwender	1	Anforderungen, Entwicklungsdokumentation, Anwenderhandbuch	Wie oben
normal (N)	ein Team für einen Kunden und einige Anwender	2-6	Projektplan, Versionen, Reviews, Testen	Buchhaltung, Lagerverwaltung
heavy (H)	viele Personen für sehr viele sehr verschiedene Anwender	10-50	Projektmanagement, Testen, Hotline, Installationen, Upgrades	Compiler, Datenbank
super heavy (SH)	sehr viele Personen programmieren etwas nie vorher Dagewesenes			Raumfahrt, Atomkraftwerk, elektronische Börse

Tabelle 2.1: Größenordnungen von Software-Engineering Projekten

Die Kategorie 'super light' symbolisiert das typische nächtliche Lust&Laune-Programm eines Informatikers. Ein Mastermind-Programm oder ein Sortieralgorithmus oder die Lösung eines kleinen mathematischen Problemchens sind Beispiele für diese Stufe. Jemand programmiert etwas für sich selbst und kaum jemand wird dieses Programmstück wieder verwenden.

In die zweite Stufe fällt der Entwicklungsauftrag an den ‘sehr gut bekannten’ Informatiker – etwa innerhalb der Familie. Beispiel: Eine Person schreibt ein Programm zur Verwaltung des bisher händisch geführten Karteikastens eines Realitätenbüros.

Der österreichische Regelfall findet sich in der dritten Kategorie wieder: Ein Team von zwei bis sechs Personen schreibt für einen Kunden mit mehreren Anwendern eine mittelgroße Software. Dabei handelt es sich genau um jene Größenordnung von Projekten, die im Rahmen der Lehrveranstaltung Software Engineering intensiver geübt wird.

Mit der Größe der Projekte steigt auch der Aufwand, um die übliche Qualität zu erreichen sehr stark. Dies wird alleine schon durch den erhöhten Personalaufwand und durch das dadurch sehr verteilte Arbeiten deutlich. Mit der Größe des Projekts steigt natürlich der Kommunikations-, Test- und der notwendige Lernaufwand.

2.2 Projektmanagement

“Projektmanagement” ist der Überbegriff für alle Tätigkeiten, die sich mit der Planung, Kontrolle, Koordination, Organisation und Beschreibung eines Projektes befassen. Ein Projektmanager muss nicht nur beim Projektstart für eine korrekte Definition und das Aufstellen eines tatsächlich durchführbaren Planes sorgen, ihm obliegt auch die Verantwortung für die Projektinfrastruktur (etwa Büroräume, Netzwerk, Einbindung in die betriebliche Organisation). Er wirkt oft bei der Auswahl der Teammitglieder mit und sorgt dafür, dass ihre Zusammenarbeit reibungslos läuft (etwa Koordination, Festlegen der Teamstruktur, Kommunikationsplanung, Konfliktlösung). Er überwacht die Einhaltung des Planes während des Projektverlaufs und setzt gegebenenfalls korrigierende Maßnahmen. Auch hat er die wesentlichen Vorgänge und Erkenntnisse für das höhere Management und zukünftige Projekte festzuhalten. Diese Aufgaben führt der Projektmanager nicht notwendigerweise alle selbst durch, er erteilt aber die entsprechenden Aufträge und ist für die Ergebnisse dieser Tätigkeiten verantwortlich. Der Begriff “Projektmanager” beschreibt im Allgemeinen also eine *Rolle* (siehe Abschnitt 2.3.2).

Das Projektmanagement konzentriert sich auf genau diese Punkte: die Planung, Kontrolle und Steuerung von Projekten. Dabei sind hier nicht nur Aufwände und benötigte Personen abzuschätzen, sondern auch die fachlichen sowie sozialen Fähigkeiten und Möglichkeiten [Hughes, Cotterell, 2002][Royce, 1998].

Wie in Abbildung 2.2 ersichtlich ist, gibt es zahlreiche Schnittstellen vom Projektmanagement (PM) zu Qualitätssicherung (QS), Software-Entwicklung (SE) und Konfigurationsmanagement (KM). Über diese Schnittstellen besorgt sich der Manager einerseits die Ist-Daten und gibt andererseits die Projektpläne weiter. Durch die Definition einer Software-Entwicklungsumgebung (SEU) werden den Entwicklern zahlreiche Vorgaben und Richtlinien bereitgestellt (z.B. Coderichtlinien oder Verpflichtung zur Verwendung von Werkzeugen).

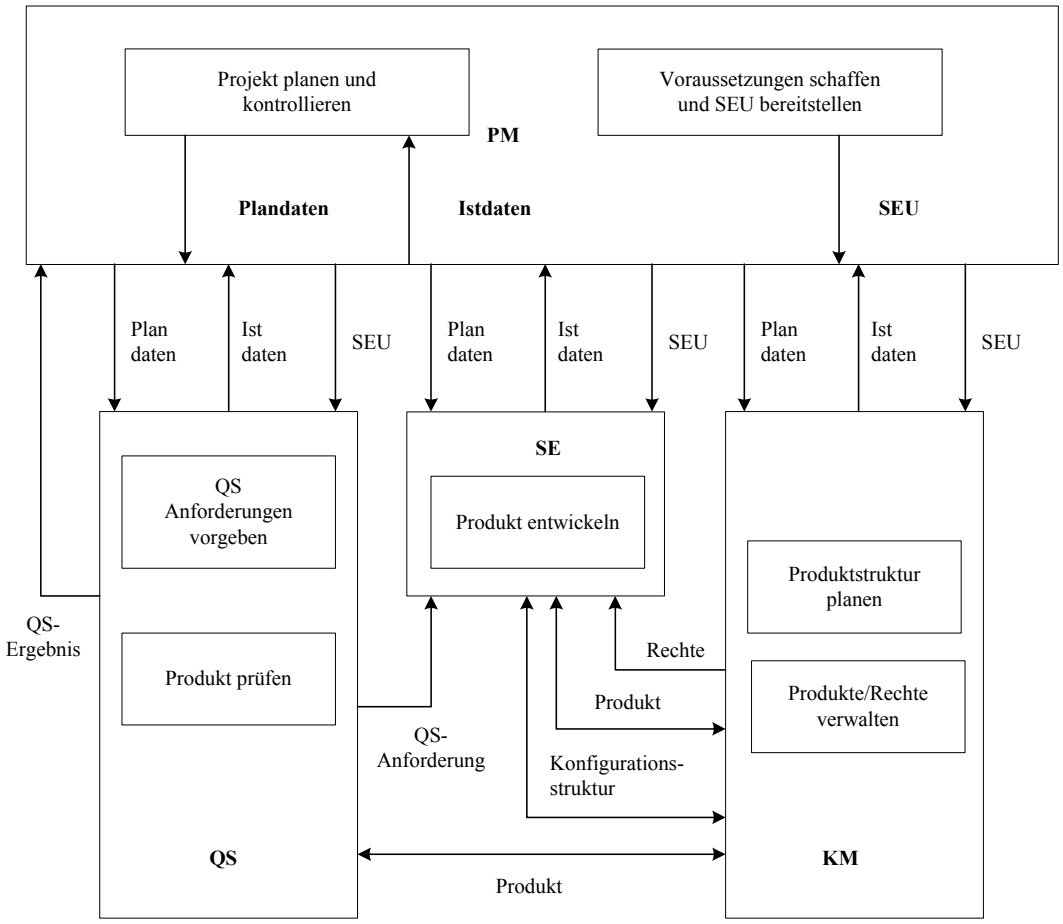


Abbildung 2.2: Schnittstellen in der Softwareentwicklung

2.2.1 Projektplanung

Die Projektplanung muss im *Projektplan* die vorhandenen Ressourcen an Geld, Personal, Zeit und Maschinen optimal einteilen. Auf Verzögerungen oder Änderungen der Arbeitsabläufe während der Projektdurchführung muss das Projektmanagement rasch und Kosten sparend reagieren können.

Besteht ein Projekt aus sehr vielen Aktivitäten und zeitlichen Abhängigkeiten zwischen den einzelnen Aktivitäten, wird die Struktur so komplex, dass eine händisch durchgeführte Einteilung der Einzelaktivitäten nicht mehr optimal sein kann und zu Verzögerungen bei nachfolgenden Aktivitäten führt. Bei großen Projekten ab ca. 100 Aktivitäten wird der Einsatz eines Netzplanprogramms obligatorisch.

- *Technische Planung*: Welche End- und Zwischenprodukte sollen erstellt werden? Auswahl eines günstigen Prozessmodells und passender Konstruktionsmethoden. Das Ziel der technischen Planung ist, den „idealen" technischen Plan zur Minimierung der Projektrisiken zu finden, bevor der Plan in Verhandlungen mit dem Management und dem Kunden den kommerziellen Realitäten angepasst wird.
- *Qualitätsplanung*: Welche Qualitätsanforderungen sollen bei den einzelnen Produkten vorhanden sein, wie kann die Qualität vom Kunden überprüft werden? Welche Risiken sollen minimiert werden? Kritische Würdigung der Methoden des technischen Planes, Hinzufügen von geeigneten Qualitätskontrollmaßnahmen.
- *Wirtschaftliche Planung*: Wie lange dauert das Projekt, wie groß ist der Aufwand an Personal und Ressourcen; welche Risiken muss ich je nach Projektumfang in Kauf nehmen? Der letzte Aspekt ist die Abschätzung des Aufwandes zur Durchführung der Aktivitäten, die bei den ersten zwei Aspekten geplant wurden, sowie der Zeitbedarf, der sich aus den Abhängigkeiten zwischen den Aktivitäten ergibt.

Eine Work Breakdown Structure (WBS) beschreibt alle Aktivitäten (z.B. Erstellung und Kontrolle von (Zwischen)Produkten, Training, usw.), für die Ressourcen (wie Zeit, Geld, Personal usw.) benötigt werden, in hierarchisch strukturierter Form. (siehe z.B. [Chapman, 1997]). Diese ergeben sich aus dem Prozessmodell, den geplanten Produkten, Methoden und Kontrollen.

Die Anzahl an Ebenen und der Detaillierungsgrad der WBS sind, wie üblich, dem Zweck angepasst zu wählen. Wesentlich ist, dass in der WBS auf keine Aktivitäten vergessen wird. Das kann zum Beispiel auch passieren, wenn Annahmen nicht gesichert sind oder nicht klar ist, woraus eine grob beschriebene Tätigkeit (z.B. "Spezifikation") im Einzelnen besteht. Motto für das Erstellen einer WBS: "Was hier fehlt, darf weder Zeit noch Geld kosten"!

Ist nun aus der WBS bekannt, was getan werden muss, müssen wir uns über die Reihenfolge klar werden, in der diese Tätigkeiten durchgeführt werden sollen. Der Netzplan folgt einer strengen Syntax: Kästchen bezeichnen Tätigkeiten, Pfeile dazwischen die Abhängigkeitsrelation. Dabei handelt es sich in der Softwareentwicklung nahezu immer um die Normalfolge. In diesem Fall bedeutet ein Pfeil von einer Tätigkeit A zu einer Tätigkeit B: *Aktivität B kann frühestens dann beginnen, wenn Aktivität A beendet ist.*[1] Auch ist der Netzplan zyklenfrei – eine durch einen geschlossenen Kantenzug beschriebene Abhängigkeit wäre nicht erfüllbar. Graphentheoretisch gesprochen ist ein Netzplan also ein azyklischer gerichteter Graph mit genau einem Anfangsknoten (Projektbeginn) und genau einem Endknoten (Projektende).

Die Tätigkeiten im Netzplan ergeben sich unmittelbar aus der WBS. Übergeordnete Aktivitäten können dabei als Meilensteine oder Subprojekte dargestellt werden. Zu den Tätigkeiten werden in der Regel zusätzliche Informationen, wie Dauer, benötigte Ressourcen, voraussichtliche Start- und Endzeit-punkte etc. festgehalten.

---

[1] Andere Folgearten sind z.B. Anfangsfolge (B kann nicht beginnen, bevor A beginnt) und Endfolge (B kann nicht beendet werden, bevor A beendet wird), mit denen wir uns hier nicht weiter beschäftigen.

2.2.2 Projektsteuerung und -kontrolle

Projektmanagement beschränkt sich – wie in der Einleitung dieses Kapitels bereits angesprochen – nicht auf die Planung zu Projektbeginn. Auch während des Projektablaufs gilt es, Maßnahmen zu setzen, um die Aktivitäten koordinieren und bei Bedarf steuernd eingreifen zu können. Das folgende Kapitel beschäftigt sich zunächst mit der Meilenstein-Trendanalyse; es folgen Versions- und Änderungs-Management und diverse Protokolle.

Meilenstein-Trendanalyse

Die Meilensteine des Projektplans markieren die Erreichung wesentlicher Stadien im Projektverlauf. Ihre tatsächliche oder geschätzte zeitliche Verteilung bietet daher einen guten und schnellen Überblick über die Terminlage des gesamten Projekts.

Bei der Meilenstein-Trendanalyse wird die Entwicklung der für die noch nicht erreichten Meilensteine geschätzten Zeitpunkte beobachtet. Dadurch lässt sich die Qualität der Schätzungen besser beurteilen; Verzögerungen im Projektablauf werden sichtbar gemacht und werden nicht erst gegen Projektende erkannt, wenn Kontrollmaßnahmen nicht mehr greifen können und der Projekterfolg in Frage gestellt wird.

Projekttagebuch und Aufwandserfassung

Teamrelevante Ereignisse werden im Projekttagebuch festgehalten. Hier findet sich eine Liste aller Besprechungen und Reviews, Integrations- und Testsitzungen etc. mit Datum, Dauer und Beteiligten. Ebenso werden zu jedem Ereignis eine Kurzbeschreibung und Informationen darüber festgehalten, wo (in welchem Dokument, z.B. Besprechungsprotokoll) Details zu finden sind.

Jeder am Projekt Beteiligte führt Buch über seine Arbeitszeit. Diese sollte aufgeschlüsselt nach bearbeiteten Dokumenten und Tätigkeiten aufgezeichnet werden, um Abweichungen gegenüber den geschätzten und geplanten Aufwänden feststellen zu können.

Besprechungsprotokolle

Die Tatsache, dass mehrere Leute beteiligt sind, macht auch kurze Besprechungen teuer. Umso wichtiger ist, dass die Ergebnisse in einer Weise festgehalten werden, die sie zur produktiven Arbeitsgrundlage macht.

Es wird festgehalten, wer in welcher Rolle teilnimmt; ebenso, wer nicht anwesend ist. Auch Datum, Dauer und Anlass werden aufgezeichnet. Welchem Zweck sollte die Besprechung dienen? Ebenso werden die zugrunde liegenden Materialien identifiziert: Über welche Dokumentversionen wird gesprochen, von welchen Voraussetzungen wird ausgegangen? In welchen Dokumenten finden sich weitere Details? In der Folge wird verzeichnet, über welche Fragen und Probleme tatsächlich gesprochen wurde.

Das Festhalten der gefällten Entscheidungen und getroffenen Ergebnisse bildet den Kern des Protokolls. Diese müssen in eine Art und Weise festgehalten werden, die es auch Nicht-Teilnehmern erlaubt, informiert zu handeln. Ebenso wichtig ist es, die aufgrund dieser Entscheidungen zu setzenden konkreten Aktionen zu bestimmen. Da auch wichtige Aufträge dazu tendieren, liegen zu bleiben, wenn sich niemand dafür zuständig fühlt, werden die für die

Durchführung Verantwortlichen notiert. Wurden gewisse Fragestellungen nicht entschieden, ist diese Tatsache unbedingt fest zu halten.

2.3 Personen & Rollen im Projekt

Personen im Projektumfeld nehmen eine Vielzahl an Aufgaben als Funktion ihrer Rollen im Projekt wahr; dementsprechend widmen sie sich verschiedenen Aufgabenstellungen und agieren mit unterschiedlichen, manchmal sogar gegensätzlichen Prioritäten.

2.3.1 Interessensgruppen in der Softwareentwicklung

Folgende Gruppen haben Interesse am Qualitätsmanagement direkt, beziehungsweise an der Qualität des Produktes:

Kunde:

- Welche funktionalen und nicht-funktionalen Anforderungen erwarte ich mir?
- Was kann/soll ich tun um das gewünschte Produkt zu erhalten?
- Erfüllt das vorgelegte Produkt meine Wünsche?

Teamleiter:

- Wo habe ich verstärkt auf Qualität zu achten?
- Erfüllen alle Teammitglieder die Qualitätsrichtlinien?
- Gibt es Probleme, welche die Qualität beeinflussen können?

Qualitätssicherer:

- Entsprechen alle Dokumente und Produkte des Teams dem Qualitätsplan?
- Wo kann die Qualität noch verbessert werden?
- Wie kann die Qualität gesteigert werden?
- Was ist zu unternehmen, um gesammeltes Wissen zu bewahren?
- Wie kann die Zusammenarbeit im Team/mit dem Kunden verbessert werden?

Technischer Architekt:

- Entspricht meine Arbeit dem Qualitätsplan?
- Was kann ich besser machen?

Tester:

- Entspricht meine Arbeit dem Qualitätsplan?
- Was kann ich besser machen?

- Wie kann ich mein Feedback an die Programmierer verbessern um die Qualität des Produktes zu steigern?

Management:

- Entspricht der Output unseren Qualitätsansprüchen?
- Wie können wir mehr Qualität erreichen?
- Wie können wir unsere Mitarbeiter zu qualitativ besserer Arbeit motivieren?
- Ist die Struktur im Unternehmen zu ändern um die Qualität zu steigern?
- Können wir uns die Steigerung der Qualität überhaupt leisten?
- Was würde es kosten?

2.3.2 Rollen im Projektmanagement

Software Engineering ist mehr als eine Lehre von Methoden, deren Syntax und deren Anwendung auf verschiedene Problembereiche. Der organisatorische Bereich von Software Engineering befasst sich mit der Aufteilung von Arbeit auf verschiedene Personen mit unterschiedlichen Rollen und Qualifikationen. Eine Rolle ist durch Kommunikationsaufgaben, Verantwortlichkeit und benötigter Kompetenz charakterisiert.

Wie verfeinert die Definition der einzelnen Rolle ist, hängt von der Anzahl der am Projekt beteiligten Personen ab. Bei Projekten mit großen Teams werden explizite detaillierte Arbeitsanweisungen benötigt, während bei kleinen Teams ein Mitarbeiter mehrere Rollen übernehmen muss. Eine grobe Zuordnung von Rollen zu konkreten Personen ergibt sich bei Betrachtung der Zeitdauer, des Aufgabenbereichs und der Tiefe der Involvierung in das Projekt.

Organisatorische Rollen begleiten, überprüfen und verwalten das Projekt vom Beginn bis zum Ende. Sie sind für Kommunikationsaufbau und -erhaltung sowie die Koordinierung der einzelnen Projektbeteiligten verantwortlich. Um immer den aktuellen Status des Projekts zu kennen, ist eine intensive, längerfristige Beschäftigung mit dem Projektverlauf notwendig. Zwei typische organisatorische Rollen sind der Projektmanager (Teamkoordinator) und der Administrator.

- *Projektmanager (Teamkoordinator)*: Zum Arbeitsauftrag des Projektmanagers gehören das Herstellen und Erhalten von Kundenkontakten, die Projektplanung, Beobachtung des Projektfortschrittes, Koordinierung von Reviews, Berichte an Kunden und Management, sowie die Projektsteuerung.
- *Administrator*: Der Administrator verwaltet das Projektarchiv, das Zusammenstellen von Unterlagen, welche an Externe weitergeleitet werden und ist für die Koordination des teamweiten Versionsmanagement, Qualitätssicherung und der Dokumente auf formalem Niveau verantwortlich.

Inhaltliche Rollen beschreiben Spezialaufgaben, die in bestimmten Projektphasen wichtig sind. Eine intensive Auseinandersetzung mit der Aufgabenstellung erfolgt nur in gewünschten, abgegrenzten Bereichen. Die wesentlichsten inhaltlichen Rollen in einem Projekt sind:

- *Domänenexperte (Analytiker)*: Voraussetzung ist ein fundiertes Wissen über den Anwendungsbereich des Kunden (die sog. Domäne). Er verfasst die Aufgabenstellung und alle aufgabenorientierten Darstellungen der Systemfunktionalität. Er überprüft die verwendungsorientierte Korrektheit der Implementierung.
- *Designer*: Er benötigt Kenntnisse über Methoden zur Umsetzung von aufgabenorientierten Dokumenten in Dokumente, welche die technische Ausführung projizieren. Er ist für die technische Korrektheit der Implementierung verantwortlich.
- *Tester*: Kenntnisse über Testmethoden, Erstellung von Testplänen und -fällen, sowie die Nutzung von Testwerkzeugen sind notwendig. Unterstützt die Erstellung der Programmierrichtlinien.
- *Werkzeugexperte*: Erstellt praktische Werkzeuge, die im Projekt von allgemeinem Nutzen sind. Überprüft die Entwicklungsumgebung auf Funktionalität und Stabilität. Ist bei der Erstellung der projektweiten Programmierrichtlinien, sowie der Datenbank- und Benutzerschnittstellenkonventionen beteiligt.
- *Dokumentierer*: Experte für Textverarbeitungs- und Zeichenprogramme, layoutet alle Dokumente und sorgt für eine einheitliche und professionelle Gestaltung der Unterlagen. Ihm obliegt die Formulierung der Dokumentationsrichtlinien.

Unter den Begriff *projektübergeordnete interne Partner* fallen Positionen, die Einfluss im strategischen Bereich haben. Sie sind typischerweise für mehrere Projekte zuständig. Sie geben Weisungen und treffen wesentliche Entscheidungen, die das Projektumfeld beeinflussen. Zu diesen Positionen zählen Management und Qualitätssicherung.

- *Management*: Es beobachtet die Projektfortschritte mehrerer Teams, korrigiert die Zielsetzung der einzelnen Projekte und versucht Konflikte zu bereinigen.
- *Qualitätssicherung*: Sind Berater für Qualitätsfragen, kontrollieren die Einhaltung der Qualitätskriterien, berichten ans Management und unterstützen bei der Review von Projektdokumenten.

  Sie entscheiden über die Qualität des ausgelieferten Produktes: Qualität im technischen Sinn ist der Grad der Erfüllung von vordefinierten Qualitätskriterien. Aus kommerzieller Sicht definiert sich Qualität aus den Bedingungen und Anforderungen, die der Kunde erstellt, unter denen er gewillt ist, das Endprodukt zu kaufen.

Eine Sonderstellung nehmen alle externen Partner ein. Sie können sowohl „wirkliche“ externe Partner sein (z.B. andere Unternehmen, Massenmarkt), aber auch demselben Unternehmen angehören (z.B. interner Dienstleister bzw. Kunde).

- *Externe Partner* sind die Auftraggeber und Verwender des herzustellenden Produktes. Sie bilden jene Instanz, die letztendlich über Qualität und Erfolg des Projektes urteilt.
- *Kunde (Käufer)*: Seine Zahlungsbereitschaft ist Richtlinie für den finanziellen Rahmen.
- *Anwender*: Beurteilt das gelieferte System. Seine Fachkenntnisse und Mitarbeit helfen bei der Aufgabenanalyse.

Abbildung 2.3 zeigt eine mögliche Positionierung und Involvierung der am Projekt Beteiligten. Anhand dieser Abbildung ist gut zuerkennen, dass externe Partner zwar mit dem Entwicklungsteam in Kommunikation stehen, jedoch mit der Entwicklung selbst nicht befasst sind.

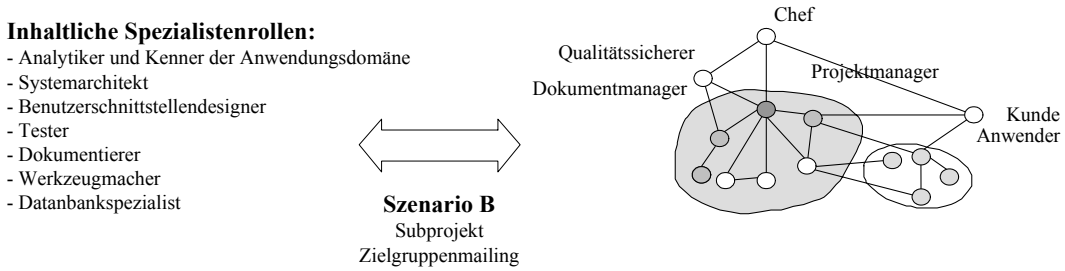


Abbildung 2.3: Involvierung der Rollen in einem Projekt

Der Projektmanager ist der Leiter im Team und vertritt das Team nach außen, d.h. gegenüber seinen Vorgesetzten, dem Kunden und dem Qualitätssicherer. Der Qualitätssicherer sollte nicht Mitarbeiter im Entwicklungsteam sein, um weitgehend objektiv die Überprüfungen durchführen zu können.

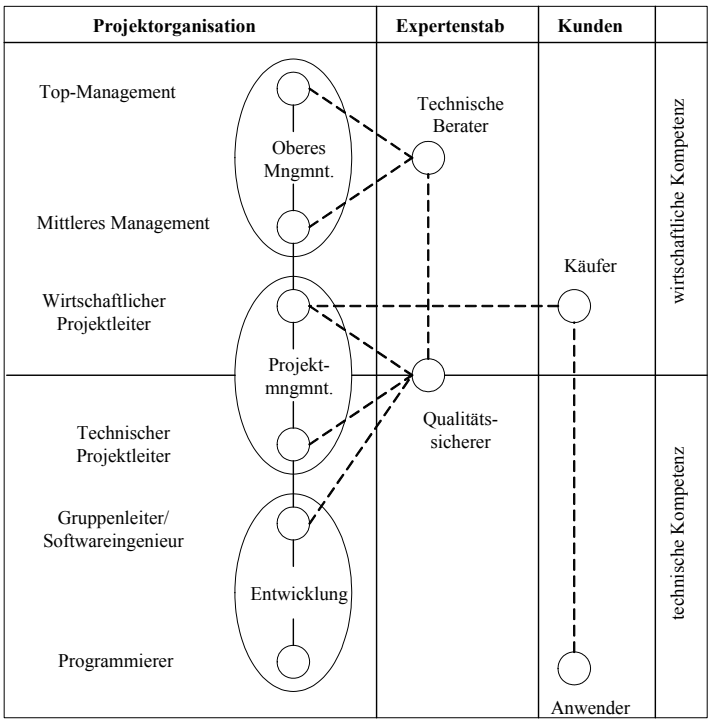


Abbildung 2.4: Rollen im Software Engineering bei großen Projekten

Abbildung 2.4 zeigt die verschiedenen Rollen hierarchisch dargestellt. Vertikal erfolgt die Aufteilung in technische und wirtschaftliche Rollen. Von links nach rechts sind Rollen auf Seite des Entwicklers, sowie des Softwarekäufers dargestellt.

2.4 Prozessmodell in der Software Entwicklung

Ein Prozessmodell bildet ein Gerüst für die Ablaufplanung eines Software-Entwicklungsprojekts. Prinzipiell "funktioniert" Software-Entwicklung auch ohne Plan, Gerüst und Konzept. In diesem Fall wird meist nach dem so genannten "Code-and-Fix"-Verfahren vorgegangen: Einfach drauflos programmieren und solange herumbasteln, bis das gewünschte Ergebnis herauskommt (oder auch nicht). Ganz abgesehen von allen anderen zahlreichen Nachteilen (wie horrende Entwicklungskosten, sollte das Produkt überhaupt jemals funktionieren) hat ein solches Vorgehen ein wesentliches Problem: Ein Projekt ohne Struktur ist nicht verwaltbar. Es kann nicht geplant und abgeschätzt werden, es können keine Meilensteine gesetzt werden, der Fortschritt kann nicht überwacht werden und man kann dem Kunden keine seriöse Zusage über Kosten und Ergebnis geben.

Software Engineering befasst sich mit der systematischen Herstellung von Software [Pressman, 2001]. Ein wesentlicher Aspekt ist dabei der geregelte Ablauf eines Entwicklungsprojektes. Zu diesem Zweck besteht ein Prozessmodell aus verschiedenen Phasen (meist Vorstudie, Analyse,

Entwurf, Codierung, Testen und Wartung) mit jeweils wohldefinierten systematischen Teilschritten und Ergebnissen (Produkte, Dokumente) sowie Anfang und Ende (Meilensteine im Projektplan zur Überprüfung des Fortschritts mittels Indikatoren) und Entscheidungsvorgaben für die Abfolge der Phasen.

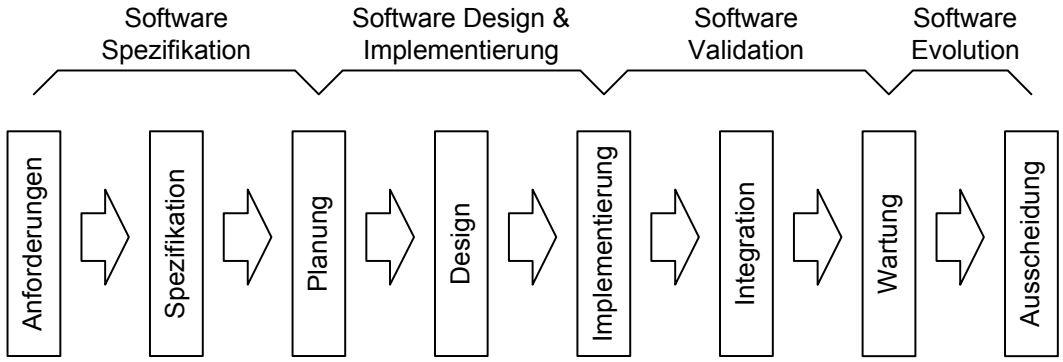


Abbildung 2.5: Software Process [Schach, 1996, Sommerville, 2001]

Gemäß Sommerville's Definition eines Softwareprozesses [Sommerville, 2001] gibt es vier grundlegende Tätigkeiten in einem Software-Engineering-Projekt. Schach behandelt diese Grundtätigkeiten noch detaillierter [Schach, 1996]. Er unterscheidet zwischen acht Phasen, die hier beschrieben werden.

Wie in Abbildung 2.5 ersichtlich, ist es möglich, die beiden Softwareprozesse einander gegenüberzustellen und dadurch eine detailliertere Ansicht zu erhalten, die immer noch die grundlegenden Arbeitsschritte beinhaltet.

2.4.1 Anforderungsanalyse

Der erste Kontakt zwischen Auftraggeber (Kunde) und Entwickler ist sehr facettenreich. Die Sicht des Kunden auf das Produkt wird von seinem eigenen Nutzen geprägt sein, er weiß nicht was er wirklich will und wie das realisiert werden kann. Der Entwickler hat seine eigene Meinung und weiß nicht genau was der Kunde will, aber er verfügt über das Know-how von einigen möglichen Lösungen, die seinen Ideen entsprechen. In der Anforderungsphase (*requirements phase*) sollte es einen Kompromiss geben, so dass klar ist, was der Auftraggeber erwarten kann und was der Entwickler verwirklichen soll. Das Abschlussdokument, die Anforderungsanalyse, beschreibt dieses Faktum. Typische Einschränkungen sind Funktionalität, Kosten, Abgabetermin, Zuverlässigkeit, etc. Während der Anforderungsanalyse erstellt das Team, welches sowohl aus Auftraggeber-Mitgliedern, als auch aus Entwicklungsteam-Mitgliedern besteht, Produkte wie die Systembeschreibung, das Begriffsverzeichnis, die Schnittstellen und Aktorenlisten, Anwendungsfalldiagramme und -beschreibungen, Prototypen etc.

Um das Einvernehmen zwischen Auftraggeber und Entwickler zu garantieren, sollte nach der letzten Fassung eine Qualitätskontroll-Methode vorgesehen werden. Die in diesem Kapitel beschriebenen Reviews und Kontrollen sind mögliche Ansätze dafür. Diese Phase kann nicht als

eigener Schritt im Prozess vorkommen, aber eine Anforderungsanalyse muss als Vorarbeit für die Erstellung der Spezifikation durchgeführt werden.

2.4.2 Spezifikationsphase

Nach der Herstellung des Anforderungsdokuments wird ein Spezifikationsdokument erstellt (*specification phase*), welches den Vertrag zwischen Auftraggeber und Entwickler begründet.

Die Spezifikation enthält neben detaillierten Informationen über die Funktionalität und über nicht-funktionalen Anforderungen an das Softwareprodukt auch das Umfeld, sowie Test- und Dokumentationsanforderungen.

Um ungewollte Änderungen in späteren Entwicklungsstufen zu verhindern, dürfen Begriffe im Text keine unterschiedlichen Interpretationen zulassen oder unklare Informationen enthalten.

Während des Schreibens der Spezifikation müssen die Entwickler auf Mehrdeutigkeiten oder unvollständige Definitionen achten.

Die Software-Qualitätskontrollgruppe muss die Spezifikation – wegen ihrer großen Bedeutung für spätere Entwicklungsphasen – überprüfen.

Sommerville betrachtet die Spezifikationsphase als den ersten notwendigen Bestandteil im Softwareprozess.

2.4.3 Planungsphase

Sommerville hat keine Planungsphase (*planning phase*) erwähnt, aber sie kann entweder der Spezifikations- oder der Designphase zugeordnet werden.

An diesem Punkt des Projekts sollte jeder (Auftraggeber und Entwickler) genau wissen, was gemäß dem Spezifikationsdokument getan werden muss. Nun hat das Projektmanagement den weiteren Entwicklungsprozess bis zum Ende des Projekts zu planen. Einige Tätigkeiten, wie beispielsweise das Testen von Codefragmenten kann nur durchgeführt werden, wenn diese vorher geschrieben wurden – manche Tätigkeiten müssen in einer bestimmten Reihenfolge – andere (wie die Programmierung und das Schreiben von Tests) müssen parallel durchgeführt werden. Der früheste Projektmeilenstein ist der Abschluss der Spezifikationen.

Die Aufgabe des Projektmanagers ist es, eine Lösung für das Projekt zu finden, welche die Dauer, die Kosten und die Ressourcen optimiert. Er muss auch dafür sorgen, dass die Softwarequalitätskontrolle zur Sicherung der guten Projektentwicklung eingesetzt wird und er muss mögliche Probleme innerhalb des Projekts berücksichtigen. Detaillierte Informationen über Projektteams, die Aufgaben eines Projektmanagers etc. können in einschlägiger Fachliteratur nachgelesen werden.

2.4.4 Design Phase

Die Spezifikationen erklären genau *was* das Produkt können muss. Das Ziel der Designphase ist es zu entscheiden, *wie* das Produkt dies macht. Die Hauptaufgabe während der Designphase ist die

Definition der internen Struktur, des Datenflusses, der Algorithmen, der Schnittstellen, etc. gemäß den in der Spezifikation dokumentierten Anforderungen. Jede Designentscheidung muss dokumentiert werden um die Ideen des Designers nachvollziehen zu können. Das kann notwendig sein, wenn eine Entscheidung zu einem "toten Ende" führt und die Designer zu einem früherem Stadium des Design zurückkehren müssen oder aus wartungstechnischen Gründen.

Für die Zukunft muss Erweiterbarkeit und Offenheit berücksichtigt werden – aber alle möglichen Anforderungen zu beachten ist beinahe unmöglich, daher muss das Designteam einen Kompromiss finden.

Da Spezifikations- und Designphase eng aneinandergekoppelt sind, ist es schwierig die Projektphase zu ermitteln. Das Design-Team erstellt das Architekturdesign, welches das Produkt in Begriffen seiner Module beschreibt und ein detailliertes Design für die Beschreibung der einzelnen Module.

Basierend auf dem Design, welches ebenfalls vom Qualitätskontrollteam freigegeben werden muss, sind die Programmierer imstande das System zu implementieren.

2.4.5 Implementierungsphase

Die Designdokumente bilden die Basis für die Implementierungsphase. Der Entwickler programmiert die einzelnen Module und testet sie gemäß dem Testplan. Weitere Dokumentation ist für spätere Projektphasen notwendig – zum Beispiel für die Instandhaltungsphase zur Rekonstruktion spezieller Realisierungen der verschiedenen Module, um Verbesserungen oder Fehlerkorrekturen durchführen zu können.

2.4.6 Integrationsphase

Wenn alle Module erzeugt und getestet sind, müssen sie zusammengebaut werden. Schach präsentiert verschiedene Strategien, um die Integration zu realisieren. Die Module können alle auf einmal oder Schritt für Schritt, aber auch von oben nach unten oder umgekehrt (entsprechend dem Modul-Verbindungsdiagramm) zusammengebaut werden.

Bei der Top-Down-Integration werden Designfehler sehr bald sichtbar. Beim Gegenstück, bei der Bottom-Up-Integration hat man zunächst den Eindruck, dass alles korrekt funktioniert, weil Designfehler erst sehr spät in der Integrationsphase entdeckt werden. Beide Strategien haben ihre Vor- und Nachteile. Die Implementierung und die Integration müssen zusammenarbeiten um die beste Lösung, eine frühe Fehlererkennung und – als Konsequenz – eine Korrektur von beispielsweise Designspezifikationen zu erreichen.

Wenn die Integration abgeschlossen ist, ist das Produkt selbst fertig für die Verwendung durch den Auftraggeber. Zu diesem Zeitpunkt startet die anspruchsvollste Phase – die Instandhaltungsphase.

2.4.7 Instandhaltungsphase, Wartung

Die Instandhaltung (*maintenance phase*) ist ein fester Bestandteil des Softwareprozesses und nicht nur eine nach der Entwicklung ausgeführte Tätigkeit um Fehler zu beseitigen. Auch die Ingenieure

der Designphase müssen Instandhaltungsaktivitäten in Bezug auf Vergrößerung und Modifizierung gemäß den Kunden-Bedürfnissen in Betracht ziehen (Anforderungen können sich ändern).

Ein kritischer Punkt in der Softwareentwicklung ist der Zeitmangel. Sowohl die Auftraggeber, als auch die Entwickler wollen das arbeitende Programm sehen. Bei Zeitdruck wird die Dokumentation vernachlässigt. Die Dokumentation umfasst sowohl das Benutzerhandbuch, als auch die Entwicklungsdokumentation und die Sourcecode-Beschreibung. Einige Zeit später, in der Instandhaltungsphase, weiß niemand wirklich warum eine spezielle Lösung verwendet wurde. Dadurch ist die Rekonstruktion spezieller Themen sehr schwierig und die Durchführung von Änderungen ebenfalls. Wenn man diesen Ansatz verfolgt müssen die Ingenieure bereits in der Design- und der Projektplanungsphase über Instandhaltungsprobleme unter spezieller Berücksichtigung der Dokumentation nachdenken.

Es ist eine der Hauptaufgaben der Qualitätskontrolle und der Projektleitung diese Vorgangsweise zu garantieren.

2.4.8 Ausmusterung

Nach jahrlanger Nutzung kann es notwendig sein, über die weitere Verwendung der Software nachzudenken. Wird die Software „verschrottet“, spricht man von der „*retirement phase*“. Es gibt vier Hauptgründe für diese Entscheidung:

- Zu viele Änderungen können der Grund für ein komplettes Redesign oder eine Neuprogrammierung aufgrund möglicher Designänderungen sein.
- Viele Änderungen, welche von anderen Aktivitäten abhängen, können den Programmstatus und das Ergebnis beeinflussen und zu einem überlasteten Programm führen. Eine kleine Modifikation in einem einfachen Modul kann zu einem komplett falschen Resultat an einer anderen Stelle führen.
- Schnelle und schlampige Änderungen ohne Aktualisierung der Dokumentation führen zu Inkonsistenz im gesamten Produkt, sodass es sicherer ist, das Programm neu zu schreiben.
- Bei einem Hardware-Wechsel ist voraussichtlich ein Redesign oder eine Neuprogrammierung des Programms notwendig um ein passendes Format zu erhalten.

Wegen dieser Gründe kann die Software veralten oder kann es sinnvoller sein, sie nicht mehr zu benutzen und ein anderes Softwareprodukt zu beginnen.

Die erwähnten Phasen begründen den Software-Lebenszyklus. Verschiedenes (wie das Zurückgehen zu früheren Phasen) ist kaum möglich – Modifikationen oder verschiedene Prozessmodelle müssen verwendet werden. Solche Modelle werden im nächsten Abschnitt beschrieben.

2.5 Zusammenfassung

Da Software Engineering eine sehr junge Ingenieursdisziplin ist, ist die Weiterentwicklung von Methoden und Vorgehensmodellen für die Projektarbeit immer noch ein primäres Ziel.

Softwareprodukte und -projekte können anhand verschiedener Merkmale typisiert werden: nach Systemtypen (kommerzielle Applikationen, Echtzeitsoftware, Web-Applikationen, Computerspiele, usw.), nach Projekttypen (Komplexität und Funktionalität, Qualitätsansprüche, Anzahl beteiligter Personen, erwartete Einsatzdauer des Produkts), nach Projektgrößen (SL, L, N, H, SH) und nach Anwendungsbereich (Versicherungen/Banken, Betriebssoftware, Medizin, Standardsoftware).

"Projektmanagement" ist der Überbegriff für alle Tätigkeiten, die sich mit der Planung, Kontrolle, Koordination, Organisation und Beschreibung eines Projektes befassen. Interessensgruppen in der Softwareentwicklung sind Kunden, Teamleiter, Qualitätssicherer, technischer Architekt, Tester und das Management. Der organisatorische Bereich von Software Engineering befasst sich mit der Aufteilung von Arbeit auf verschiedene Personen mit unterschiedlichen (organisatorischen oder inhaltlichen) Rollen und Qualifikationen.

Ein wesentlicher Aspekt der systematischen Herstellung von Software ist der geregelte Ablauf eines Entwicklungsprojektes. Dieser kann durch den Einsatz eines Prozessmodells, welches ein Gerüst für die Ablaufplanung bietet, erreicht werden. Mögliche Phasen eines Prozessmodells sind Anforderungsanalyse, Spezifikations-, Planungs-, Design-, Implementierungs-, Integrations-, Wartungs- und Ausmusterungs-Phase.

2.6 Literaturreferenzen

[Boehm, 1998] Boehm, Barry W: „*A Spiral Model of Software Development and Enhancement*", 1998, http://www.sce.carleton.ca/faculty/ajila/4106-5006/Spiral%20Model%20Boehm.pdf

[Chapman, 1997] Chapman, James R.: „Work Breakdown Structure (WBS)" http://www.hyperthot.com/pm_wbs.htm

[Hughes, Cotterell, 2002] Hughes, Bob; Cotterell, Mike: „*Software Project Management*" 3rd Edition, McGraw-Hill, 2002, ISBN: 007709834X

[Pressman, 2001] Pressman, Roger S.: „*Software Engineering: A Practitioner's Approach"*, 5th Edition, MacGraw-Hill, 2001, ISBN: 0072496681

[Royce, 1998] Royce, Walker: „*Software Project Management: A Unified Framework*", Addison Wessley, 1998, ISBN: 0201309580

[Schach, 1996] Schach, Stephen R.: „*Classical and Object-Oriented Software Engineering*", 3rd Edition, Vanderbilt University, 1996, ISBN: 0256182981

[Shaw, 1996] Shaw, Mary; Garlan, David: "*Software Architecture: Perspectives on an Emerging Discipline*", Prentice Hall, 1996, ISBN 0131829572

[Sommerville, 2001] Sommerville, Ian: „*Software Engineering*", 6th Edition, Addison Wesley, 2001, ISBN: 020139815X

[Zuser et al., 2001] Zuser, Wolfgang; Biffl, Stefan; Grechenig, Thomas; Köhle, Monika: „*Software Engineering mit UML und dem Unified Process*", Pearson Studium, 2001, ISBN: 3827379272

2.7 Übungsaufgaben

1. Wie lauten die 4 angegebenen Typen von Softwaresystemen? Gibt es außer diesen noch weitere Typen?
2. Ein Projektmanager erweitert aufgrund des extrem hohen Zeitdrucks sein Projektteam um 10 weitere Mitarbeiter. Anstatt, dass der vorgegebene Termin mit dem verstärkten Team erreicht wird, verspätet sich die Fertigstellung des zu entwickelnden Produktes. Woran kann das liegen?
3. Wodurch unterscheiden sich verschieden große Software-Projekte (Größenordnungen: super light, light, normal, heavy, super heavy?
4. Welche zwei planungsunterstützenden Hilfsmittel gibt es, um sich einen Überblick über die notwendigen Aktivitäten und deren Reihenfolge zu beschaffen?
5. Stellen Sie sich vor, Sie sind Projektmanager und stehen ganz am Anfang eines neuen Software-Projekts. Überlegen Sie sich grob welche Tätigkeiten notwendig sind, um das gewünschte Produkt fertig zu stellen. Fassen Sie diese Tätigkeiten in einer Work Breakdown Structure zusammen. Sie können dabei allgemeine Tätigkeiten in detaillierte Aktivitäten aufteilen, um eine hierarchische Übersicht über die Tätigkeiten zu bekommen.
6. Stellen Sie die Abhängigkeiten der oben entwickelten Tätigkeiten in einem Netzplan dar.
7. Welche Methode gibt es, um den Ist-Zustand des Projektfortschritts mit dem Soll-Zustand zu vergleichen?
8. Welche Rollen gibt es typischerweise in einem Software-Projekt?
9. Ein typischer Software-Prozess besteht aus 8 Prozessschritten. Wie lauten sie und was ist jeweils ihr Inhalt?