Anwendungsentwicklung mit MSF

Bei Entwurf und Entwicklung von Unternehmensanwendungen müssen heute viele verschiedene Anforderungen erfüllt werden. Jede Entwicklungsentscheidung, die gefällt wird, um eine Anforderung zu erfüllen, wirkt sich auf viele andere Anforderungen aus. Oft sind diese Auswirkungen nicht vorhersehbar oder schwer nachzuvollziehen. Wenn auch nur eine Anforderung nicht erfüllt werden kann, kann dies das Scheitern des gesamten Projekts bedeuten. Häufig müssen auch eine Vielzahl von Anwendungsanforderungen gegeneinander abgewogen werden wie

- die Geschäftsziele,
- der gewünschte Fertigstellungstermin,
- das Budget,
- die Anzahl der Mitarbeiter, die für Entwicklung, Test und Pflege eingesetzt werden,
- die Wichtigkeit von Leistung und Benutzerfreundlichkeit,
- die damit eingesetzte Hardware
- der Einsatzbereich,
- die Sicherheitsanforderungen,
- die Einsatzdauer,
- die Anzahl der Benutzer, die gleichzeitig damit arbeiten.

Ohne eine systematische Vorgehensweise zum Erfassen der komplexen, zusammenhängenden und oftmals konkurrierenden Anforderungen gestaltet sich der Einstieg schwierig.

Die nachfolgende Beschreibung ist eine Zusammenfassung des Buches

Anforderungsanalyse und Anwendungsdesign
Praktisches Selbststudium zu den Grundlagen der Entwicklung von verteilten Anwendungen
Microsoft Press Deutschland
ISBN 3-86063-906-4
776 Seiten

Autor Christoph Ascherl
Datei Anwendungsentwicklung mit MSF.doc
Datum 8. Juli 2003

LITTLE softwarehouse GmbH
Zentnerstr. 19, D-80789 München
www.littlesoft.com

1 Entwicklungsteammodell

Da in jedem Projekt viele Aufgaben erledigt und verschiedene Gesichtspunkte berücksichtigt werden müssen, kann man die Erfolgschancen eines Projekts verbessern, indem man die Aufgaben, die wichtig für den Erfolg sind, ganz bestimmten Teamrollen überträgt. Für jede Rolle werden klar definierte Aufgaben festgelegt. Die einzelnen Rollen sind zueinander gleichberechtigt!

In der Regel wird jeder Rolle eine Reihe von Mitarbeitern zugewiesen, welche die damit verbundenen Aufgaben ausführen. Dabei übernimmt eine Person die Funktion des Teamleiters und fungiert für diese Rolle dem Projektteam als Ansprechpartner nach außen.

Hinweis: Wird eine Aufgabe mehreren Mitarbeitern zugeteilt, besteht die Gefahr, dass sich niemand dafür verantwortlich fühlt.

Rolle	Verantwortungsbereich
Produkt-Management	Kundenerwartung feststellen und Umsetzung überprüfen
	Kommunikation Kunde – Projektteam
Programm-Management	Koordination des Entwicklungsprozess
	Projektzeitplan, Meilensteine definieren
Entwicklung	Technologische Beratung des Projektteams
	Architekturentwurf
	Funktionsspezifikation erstellen
	Codierung
Benutzerschulung	Interessensvertretung der Benutzer (!= Kunde)
	Kommunikation zwischen Benutzer und Projektteam
	Schulung der Endanwender
Test	Erkennung von Schwachstellen vor Auslieferung
	Änderungskontrolle (Fehlerdatenbank)
Logistik-Management	Installation
	Produktsupport
	Helpdesk

Tabelle 1 Teamrollen und Verantwortungsbereiche in der Anwendungsentwicklung

2 Entwicklungsprozessmodell

Das Entwicklungsprozessmodell umfasst vier Hauptphasen, die jeweils mit einem Hauptmeilenstein abgeschlossen werden. Bei Erreichen eines jeden dieser Hauptmeilensteine muss der Auftraggeber entscheiden, ob der Auftrag weiter gemacht oder das Projekt gestoppt wird. Die entsprechende Frage lautet dann: „Bietet das Produkt genügend Vorteile, um die Kosten zu rechtfertigen?“

Die vier Phasen sind

- Zielvorstellung gewinnen
- Planung
- Entwicklung
- Stabilisierung

2.1 Phase 1: Zielvorstellung gewinnen

Die Phase „Zielvorstellung gewinnen“ dient der Ideenfindung. Daher wird diese Phase auch „Ideenfindungsphase“ genannt. Die Frage dieser Phase lautet: „Können wir mithilfe von Technologie eine Lösung für diese Geschäftsanforderung finden und wenn ja, wie?“

Die während dieser Phase formulierte Zielvorstellung muss allen an der Anwendungsentwicklung beteiligten Mitarbeitern immer bekannt sein.

2.1.1 Ziele

- Beiderseitiges Verständnis der geschäftlichen Anforderungen
- Erwartungen des Kunden ermitteln
- Einschätzung der Rahmenbedingungen
- Risikomanagement
  - Ermittlung von Risikoquellen
  - Risikoanalyse
    - Wahrscheinlichkeit des Auftretens eines Risikos, bewertet mit 1 (niedrig) bis 3 (hoch)
    - Auswirkung bei Auftreten des Risikos, bewertet mit 1 (niedrig) bis 5 (höchste Auswirkung)
  - Risikomaßnahmenplanung
  - Risikoüberwachung

2.1.2 Verantwortungsbereiche

Nachfolgende Tabelle zeigt einige Beispiele der spezifischen Verantwortungsbereiche der Teamrollen innerhalb dieser Phase. Die Leitung der jeweiligen Teamrolle ist dafür verantwortlich, dass diese Aufgaben ausgeführt werden und kommuniziert mit den übrigen Mitgliedern des Projektteams.

Rolle	Verantwortungsbereich
Produkt-Management	Erarbeitung des Zieldokuments
	Umgang mit Kundenerwartungen
	Einbindung des Kunden in die Prototypenentwicklung
	Risikomanagement
Programm-Management	Erarbeitung der Entwicklungsziele
	Beschreibung des Lösungskonzepts
	Entwurf der Projektstruktur
	Risikomanagement

Entwicklung	Prototypenentwicklung
	Entwurf der Entwicklungsoptionen
	Auswirkungen definieren
	Risikomanagement
Benutzerschulung	Ermittlung der Anforderungen hinsichtlich der Benutzerunterstützung
	Umgang mit Benutzererwartungen
	Einbindung des Benutzers in die Prototypenentwicklung
	Risikomanagement
Test	Entwicklung von Teststrategien
	Festlegung von Akzeptanzkriterien
	Entwicklung eines Fehlerüberwachungssystems
	Entwicklung eines Risikomanagementsystems
	Risikomanagement
Logistik-Management	Ermittlung der Auswirkungen, die sich durch die Verteilung ergeben
	Ermittlung der Supportanforderungen
	Risikomanagement

Tabelle 2 Teamrollen und Verantwortungsbereiche in der Ideenfindungsphase

2.1.3 Ergebnisse

- Dokument über das zu entwickelnde Produkt, die durch dieses erfüllten Anforderungen, die enthaltenen Funktionen sowie der vorläufige Zeitplan.
- Projektstrukturdokument, das Verantwortlichkeiten und Teams festlegt, sowie Verwaltung, Änderungsmanagement und Statusberichte beschreibt.
- Risikobewertungsdokument
- Ggf. Prototyp für konkrete, visuelle Darstellung der Produktziele, der nicht wiederverwendet wird, um ihn so einfach wie möglich zu halten.

Hinweis: Jedes Ergebnis sollte das Ziel haben, als effizientes Kommunikationswerkzeug zu dienen. In diesem Zusammenhang muss ein Ergebnis nicht unbedingt ein gedrucktes Dokument sein, sondern kann die jeweils passende Form annehmen (Dokument, Diagramm, Anwendung, Maskenabbildung, e-Mail etc.), solange diese die Kommunikation erleichtert.

2.2 Phase 2: Planung

Die Planungsphase ist das eigentliche Herzstück eines Entwicklungsprozesses, die der Architektur der Anwendung die oberste Priorität einräumt. Diese wird die Grundlage für die Erstellung des Codes und der ausführbaren Dateien für die Anwendung.

In der Planungsphase wird die folgende Frage gestellt: „Was ist, realistisch betrachtet, erforderlich, um die in der Ideenfindungsphase erarbeitete Zielvorstellung zu verwirklichen?“. Das Ergebnis ist ein detaillierter Plan, mit dem sowohl Kunde als auch Projektteam einverstanden sind.

2.2.1 Ziele

- Festlegung des gleichseitigen Variablendreiecks „Zeit, Ressourcen und Leistungsmerkmale“. Die Kunst besteht darin, dieses Gleichgewicht in der Entwicklungsphase beizubehalten. Nehmen z.B. die Funktionen zu, müssen als Ausgleich mehr Zeit oder Ressourcen zur Verfügung gestellt werden.

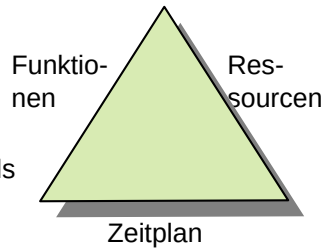

Abbildung 1
Das Variablendreieck

- Übereinstimmung mit dem Kunden, was genau entwickelt wird, um die geschäftlichen Anforderungen zu erfüllen.
- Priorisierung der geforderten Funktionen.
- Abschätzung der Projektdauer.
- Bestimmung der Architektur und der Entwicklungsumgebung.
- Überprüfung der Risikoabschätzung.
- Definition untergeordneter Meilensteine und Zwischenergebnisse.

2.2.2 Verantwortungsbereiche

Obwohl das Team gemeinsam an der Erreichung des am Ende der Planungsphase liegenden Hauptmeilensteins arbeitet, konzentrieren sich die einzelnen Rollen im Team während dieser Phase auf unterschiedliche Bereiche. Jedes Team ist für die Erstellung eines entsprechenden Aktionsplans und Zeitplans verantwortlich.

Rolle	Verantwortungsbereich
Produkt-Management	Steuert den Prozess der Erfassung der Anforderungen sowie des konzeptionellen Entwurfs. Arbeitet an Kommunikationsplan und Zeitplan.

Programm-Management	Steuert den Gesamtentwurf, speziell den logischen Entwurf. Fertigt einen Entwurf der Funktionsspezifikation an. Übernimmt die Gesamtleitung und steuert Prozess der Entscheidung, ob das Team Pläne und Zeitplan erfüllen kann.
Entwicklung	Steuert den physischen Entwurf der Funktionsspezifikation. Ermittelt den für die Erstellung und Stabilisierung des Produkts erforderlichen Zeit- und Ressourcenaufwand. Entwickelt, falls erforderlich, ein Modellsysteme für den Konzeptbeweis.
Benutzerschulung	Analysiert die Bedürfnisse der Benutzer. Erarbeitet Strategien zur Unterstützung der Systemleistung. Bewertet die Verwendbarkeit des fertigen Entwurfs. Schätzt den zur Entwicklung von Systemen zur Benutzerunterstützung erforderlichen Zeit- und Ressourcenaufwand. Testet die Eignung und Verwendbarkeit aller Benutzeroberflächenkomponenten.
Test	Bewertet den Entwurf, um zu ermitteln, ob die Funktionen korrekt getestet werden können. Erarbeitet einen Testplan und einen Zeitplan für Funktionstests. Erarbeitet Methoden und Maßstäbe für die Rückverfolgung von Fehlern. Entwickelt Teststrategien.
Logistik-Management	Bewertet Einsetzbarkeit, Benutzerfreundlichkeit, Supportfähigkeit und Gesamtkosten des Entwurfs. Legt einen Zeitplan für Installations- und Supportpläne fest.

Tabelle 3 Teamrollen und Verantwortungsbereiche in der Planungsphase

2.2.3 Ergebnisse

- Funktionsspezifikation, die das zu entwickelnde Produkt und die Anforderungen beschreibt, die dieses erfüllt.
- Projektplan, der zeigt, wie das Team das Projekt auszuführen plant und weitere untergeordnete Meilensteine und zugehörige Ergebnisse enthält, die erzielt werden sollen.
- Projektzeitplan, der den erforderlichen Zeitrahmen für die einzelnen Meilensteine und Ergebnisse beschreibt.

- Risikobewertungsdokument, das potenzielle Risiken für das Projekt auflistet und beschreibt, wie das Team mit dem jeweiligen Risiko umgehen möchte.
- Ggf. Modellsystem für den Konzeptbeweis zur Demonstration und Überprüfung der Entwurfsdetails.

2.2.4 Entwurfsprozess

Die Aufgabe des Entwurfs ist hauptsächlicher Bestandteil der Planungsphase. Der Entwurfsprozess dient dem konzeptionellen Entwurf der Anwendung unter spezieller Berücksichtigung von n-Schichten Anwendungen (auch „n-tier“ – siehe Anhang, Kapitel 4.1 „Mehrschichtenanwendungen“)

Der Entwurfsprozess umfasst drei Arten von Entwurfsansätzen: den konzeptionellen, den logischen und den physischen Ansatz. Jeder Abschnitt des Prozesses betrachtet den Anwendungsentwurf aus einer anderen Perspektive und liefert eine eigene Definition der Lösung.

Entwurfsansatz	Perspektive	Maßnahme
Konzeptionell	Betrachtet das Problem aus Perspektive von Benutzer und Unternehmen	Definiert Problem und Lösung als Szenarien
Logisch	Betrachtet die Lösung aus der Perspektive des Projektteams	Definiert die Lösung als kooperierende Dienste
Physisch	Betrachtet die Lösung aus der Perspektive der Entwickler	Definiert Dienste und Technologien der Lösung

Tabelle 4 Entwurfsansätze für die drei Teile des Entwurfsprozesses

Die Ergebnisse jedes Abschnitts werden als Ausgangsinformationen des nachfolgenden Abschnitts verwendet. Trotzdem können die drei Abschnitte parallel verlaufen. Durch iterative Zyklen werden die einzelnen Abschnitte immer wieder überprüft und überarbeitet.

Hinweis: Natürlich müssen die Aufgaben aus rein praktischen Erwägungen zu einem gewissen Grad in einer festgelegten Reihenfolge ausgeführt werden. Der konzeptionelle Entwurf muss vor dem logischen und dieser vor dem physischen Entwurf begonnen werden.

Konzeptioneller Entwurf

Der konzeptionelle Entwurf generiert *konkrete* Szenarien, die vollständige und präzise Anforderungen wiedergeben, indem sie Kunde, Benutzer und Projektauftraggeber berücksichtigen. Dazu müssen Anwendungsfälle recherchiert werden.

Ein Szenario ist eine bestimme Instanz eines Anwendungsfalles und zeigt entweder das Ist- oder das Sollverhalten mit den Informationen *Kontext* (z.B. Kultur, Firmenpolitik, Standards, Vorschriften), *Arbeitsablaufprozess*, *Aufgabenabfolge* und *physische Umgebung* (z.B. arbeitseinschränkende oder –unterstützende Bedingungen, Schemata oder Grundrisse des Arbeitsbereiches oder der Computer und des Netzwerkes, Standortkarten etc.).

Der konzeptionelle Entwurf verfolgt folgende Ziele:

- Ein Entwurf, der auf echten Daten von Kunde und Benutzern basiert.
- Ein schlüssiges, integriertes Bild des Produktes.
- Genaues Verständnis, wie sich das Produkt auf Unternehmen und Benutzer auswirkt.
- Gruppenkonsens in Bezug auf den Entwurf.
- Synchronisierung mit der Unternehmensarchitektur.

Zu den Aufgaben gehört daher:

- Ermittlung der wichtigsten Geschäftsprozesse und –abläufe.
- Festlegung von Prioritäten für Prozesse und Aktivitäten.
- Identifizierung von Benutzern und Erstellung von Profilen.

Logischer Entwurf

Der logische Entwurf formuliert anhand der konzeptionellen Entwurfsszenarien ein *abstraktes* Modell der Lösung.

Ergebnis des logischen Entwurfs sind eine Reihe von Geschäftsobjekten und die zugehörigen Dienste, Attribute und Beziehungen, sowie ein allgemeiner Entwurf für die Benutzeroberfläche und ein logischer Datenbankentwurf. Die Einteilung in die Schichten einer n-Schichtenanwendung erfolgt bereits hier.

Ein *Dienst* ist eine Einheit der Anwendungslogik, durch die ein Vorgang, eine Funktion oder eine Umwandlung implementiert wird. Es wird dabei nichts über seine Komplexität ausgesagt.

Ein *Geschäftsobjekt* kapselt Dienste und verwendete Daten und dient zur Strukturierung der Lösung und zur Reduzierung ihrer Komplexität. Geschäftsobjekte repräsentieren die in den Szenarien beschriebenen Personen oder Gegenstände. Diese Objekte werden nun zu „Ankerpunkten“ für die Attribute und Beziehungen.

Attribute sind die Elemente eines Objekts, die dem Unternehmen bekannt sein müssen und die es überwachen muss. Bei Attributen handelt es sich um die Definitionen von Datenwerten, die von einem Objekt gespeichert werden. Jede Instanz eines Objektes verwaltet eine eigene Reihe von Werten, die auf der zugehörigen Definition basieren.

Eine *Beziehung* ist eine logische Zuordnung von Objekten. Zur Festlegung eines effektiven Entwurfs und zur Zusammenstellung der Systemkomponenten ist es erforderlich, Beziehungen zu erkennen. Sie beschreiben, wie Objekte miteinander verwandt oder verknüpft sind.

Nach Identifikation und Formulierung der Geschäftsobjekte erfolgt ihre Aufteilung auf Benutzerschicht, Geschäftsschicht und Datenschicht mit dem Ziel Flexibilität, Skalierbarkeit und Wiederverwendbarkeit zu erreichen.

Physischer Entwurf

Festlegung von *konkreten* Lösungsdetails auf die Ergebnisse des logischen Entwurfs unter Berücksichtigung praxisbezogener technologischer Einschränkungen, einschließlich Überlegungen zu Implementierung und Systemleistung.

Der physische Entwurf ist definiert als der Prozess zur Beschreibung der Komponenten, Dienste und Technologien der Lösung aus der Entwicklungsperspektive.

Ergebnis des physischen Entwurfs sind eine Reihe von Komponenten, der Entwurf einer Benutzeroberfläche für eine bestimmte Plattform sowie ein Entwurf der physischen Datenbank.

2.2.5 Funktionsspezifikation

Die Funktionsspezifikation ist das wichtigste Dokument, das am Ende der Planungsphase geliefert werden muss. Dabei handelt es sich um eine Reihe von Spezifikationen für das Produkt. Das Dokument dient als Vertrag zwischen Kunde und Projektteam. Aus der Funktionsspezifikation resultieren die verbleibenden Ergebnisse, welche die Planungsphase zu liefern hat: Projektplan, Projektzeitplan und überarbeitetes Risikobewertungsdokument.

Ziel bei der Erstellung einer Funktionsspezifikation ist der Konsens aller am Projekt beteiligter Personen.

Der Inhalt der Funktionsspezifikation umfasst folgende Komponenten:

Komponente	Beschreibung
Zielformulierung	Zielsetzung, Rechtfertigung und wichtige, allgemeine Beschränkungen für das Produkt. Basiert auf dem Zieldokument aus der Ideenfindungsphase (Phase 1: Zielvorstellung gewinnen).
Entwurfsziele	Was das Team mit dem Produkt erreichen möchte. Bei der Entwicklung werden diese Ziele zur Entscheidungsfindung bei Fragen wie Systemleistung, Zuverlässigkeit, Pünktlichkeit und möglicherweise Benutzerfreundlichkeit und Zugriff verwendet. Diese Ziele wurden ursprünglich in der Ideenfindungsphase erarbeitet (Phase 1: Zielvorstellung gewinnen).
Anforderungen	Anforderungen, die Kunde, Benutzer und andere Interessengruppen an das Produkt stellen. Die Anforderungen sollten mit Prioritäten versehen werden. Miteinander in Konflikt stehende Anforderungen müssen auf irgendeine Weise gelöst oder gegeneinander aufgewogen werden.
Beschreibung des Einsatzbereiches	Wann das Produkt von wem verwendet werden soll. Dies ist eine allgemeine Zusammenfassung der Verwendungsszenarien, die während des Entwurfsprozesses definiert wurden.

Funktionen	Die einzelnen Funktionen des Produkts. Eine mit Prioritäten versehene Liste der einzelnen Produkteigenschaften, die z.B. auch die potenzielle Benutzeroberfläche, Anwendungsnavigation und detaillierte Funktionen umfasst.
Abhängigkeiten	Wovon das Produkt abhängig ist. Eine Beschreibung externer Einheiten, von denen das Produkt abhängig ist, einschließlich allgemeiner (z.B. Schnittstellen mit Unternehmenssystemen) und detaillierter Aspekte (z.B. gemeinsam genutzte Komponenten).
Zeitplan	Definition des Zeitplanes. Die Zusammenfassung des Projektplanes unter Angabe der wichtigsten untergeordneten Meilensteine, Ergebnisse sowie des Lieferdatums für das fertige Produkt.
Risiken	Beschreibung der Risiken. Eine Liste der Risiken, für die Transparenz nach außen erforderlich ist.
Anhänge	Verbleibende Informationen. Eine Zusammenstellung der Entwurfsergebnisse, die das Team zur Entwicklung der Funktionsspezifikation verwendet hat.

Tabelle 5 Die Wichtigsten Komponenten der Funktionsspezifikation

2.2.6 Projektplan

Der Projektplan beschreibt, wie das Produkt erstellt werden soll, indem er detaillierte Pläne aller Teammitglieder zusammenfasst. Das Team verwendet diese detaillierten Zieldefinitionen dann zur Synchronisierung der Aufgaben für die verbleibende Zeit des Projektes.

Der Projektplan hat folgende Ziele:

- Konsolidierung der Arbeitspläne für Funktionsteams und Rollen.
- Beschreibung der Aufgabenausführung für Funktionsteams und Rollen.
- Synchronisierung der Pläne im gesamten Team.

Der Projektplan sollte die in Tabelle 6 aufgeführten Punkte umfassen.

Plan	Beschreibung
Entwicklung	Beschreibt, wie die Entwicklung die in der Funktionsspezifikation definierten Komponenten erstellen wird. Dazu werden z.B. Tool, Methodik, bewährte Verfahren, Ereignisabfolgen und Testmethoden beschrieben.

Test	Beschreibt eine Teststrategie, die einzelnen zu testenden Bereiche, sowie die hierfür benötigten Ressourcen (Hardware und Mitarbeiter).
Benutzerschulung	Beschreibt eine Strategie und einen Plan zur Entwicklung der erforderlichen Schulungsunterlagen.
Benutzersupport	Beschreibt eine Strategie und Details zur Entwicklung von Hilfsmitteln, die der Benutzer benötigen wird (z.B. Assistenten und Benutzerhandbücher).
Kommunikation	Beschreibt die Marketingstrategie und die Werbemaßnahmen.
Einsatz	Beschreibt eine Strategie und umfasst einen detaillierten Plan zur Vorbereitung der Endbenutzer und Einsatzmitarbeiter vor und während der Inbetriebnahme.

Tabelle 6 Komponenten des Projektplanes

2.2.7 Projektzeitplan

Das Team kann den Zeitplan erst dann aufstellen, wenn es die zu entwickelnden Funktionen kennt und weiß, wie das Team diese liefern möchte.

Der Projektzeitplan umfasst die folgenden Details:

- Entwicklungszeitplan
- Interne und externe Versanddaten für das Produkt
- Testzeitplan
- Schulungszeitplan
- Zeitplan für den Benutzersupport
- Kommunikationszeitplan
- Einsatzzeitplan

Bei der Erarbeitung von Plänen und Zeitplänen sollte das Team bestimmte Prinzipien immer im Auge behalten. Die folgenden vier Prinzipien sollten zur Erstellung von realistischen, erreichbaren Zeitplänen beitragen.

Bottom-Up-Planung

Die Mitarbeiter, die bestimmte Aufgaben ausführen werden, sollten diese auch planen. Dieses Prinzip sorgt nicht nur für präzisere Schätzungen, sondern fördert auch die Akzeptanz des Zeitplanes innerhalb des Teams.

Festes Lieferdatum

Die Festlegung des Lieferdatums eliminiert Ausreden und begrenzt Entscheidungen über Zugeständnisse auf Ressourcen und Funktionen. Auf diese Weise wird die Bedeutung einer pünktlichen Fertigstellung unterstrichen. Die Festlegung dient auch als Antrieb für Entscheidungen darüber, welche Komponenten in welchem Umfang entwickelt werden sollen.

Risikogesteuerte Zeitplanung

Dies bedeutet, die schwierigsten, risikoreichsten Aufgaben zuerst auszuführen. Dieser Ansatz bietet erhebliche Vorteile:

- Die größten Risiken sind in der Regel die mit den meisten Unbekannten. Wenn diese zuerst bearbeitet werden, gewinnt das Team mehr Zeit zur Erarbeitung der Risikobegrenzung, wo diese sinnvoll ist.
- Greift das Team zu einem früheren Zeitpunkt ein entscheidendes Risiko an und erkennt, dass eine Beseitigung des Risikos nicht möglich ist, kann das Projekt viel früher und mit minimaler Ressourcenverschwendung eingestellt werden.
- Der Prozess, in dessen Verlauf ein Verständnis der Kundenwünsche gewonnen und erklärt wird, welches die größten Risiken sind, ermöglicht einen besseren Umgang mit Kundenerwartungen.
- Die Begrenzung entscheidender Risiken (d.h. Risiken, die ernsthaft genug sind, um das Projekt zu beenden) erfordert oft die Entwicklung von Modellsystemen für den Konzeptbeweis im Laufe des Planungsprozesses, mit denen Wahrscheinlichkeit oder Auswirkungen der Risiken bewiesen bzw. widerlegt werden können.

Eine risikogesteuerte Zeitplanung kann nur dann betrieben werden, wenn die Risiken mithilfe eines Risikomanagements priorisiert wurden.

Zeitplanung für eine unsichere Zukunft

Dieses Prinzip basiert auf der Überlegung, dass die Zukunft unsicher ist und das Team daher Pläne aufstellen muss, die an unvorhergesehene Ereignisse angepasst werden können. Drei Methoden können zur Erstellung solcher Pläne beitragen.

Hinzufügen eines Zeitpuffers

Das Programm-Management fügt am Ende des Zeitplans einen Zeitpuffer hinzu, um Spielraum für ein späteres Lieferdatum zu schaffen. Dieser ist zwar oft ein Streitpunkt und wird vom Management nicht gern gesehen, er ist jedoch ein wichtiger Teil einer effektiven Zeitplanung. Die Länge des Zeitpuffers ist direkt abhängig vom Vertrauen, das das Programm-Management in die Schätzungen der Teammitglieder setzt.

Das interne Lieferdatum wird durch Bottom-Up-Planung ermittelt. Dieses Datum plus der Zeitpuffer ergibt das extern mitgeteilte Lieferdatum.

Der Zeitpuffer liegt im Verantwortungsbereich des Programm-Managements. Die Einzelressourcen müssen sich ausschließlich auf die von ihnen selbst geschätzten Datumsangaben konzentrieren.

Verwendung von untergeordneten Entwicklungsmeilensteinen

Diese Meilensteine sind Alpha- und Betaversionen innerhalb eines Projekts. Die Aufteilung von Projekten in kleinere Abschnitte bietet dem Team zusätzliche Indikatoren, anhand derer es früher und öfter erkennen kann, wie das Projekt voran kommt. Das Projekt wird dadurch auch stabiler und der Zeitplan berechenbarer.

Die Verwendung von „Übungs“-Versionen mit eigener Stabilisierungsphase ermöglicht es dem Team, den Versand zu üben und ist darüber hinaus ein Indikator für den Zustand des Projektes.

Verwendung von diskreten Aufgaben

Das Team kann das Projekt innerhalb des geplanten Rahmens halten, indem es für kurze Intervalle sichtbare Ergebnisse definiert. Dazu sollten die Produktfunktionen in zugehörige Aufgaben aufgeteilt werden, die klar definiert sind und über einen Ausgangs- und Endpunkt, ein einziges Ergebnis und einen einzigen Verantwortlichen verfügen. Der Umfang sollte sich auf Aufgaben beschränken, die höchstens zwei Wochen erfordern.

Die einzelnen Aufgaben sollten dann im Laufe des Projekts verfolgt und die Zeitpläne entsprechend angepasst werden.

2.3 Phase 3: Entwicklung

Die Entwicklungsphase ist die dritte Phase des Prozessmodells. Während der Entwicklungsphase konzentriert sich das Team auf die Produktausführung. Hier wird der vollständige Anwendungscode implementiert und zunächst für das Projektteam freigegeben.

Spezifische untergeordnete Meilensteine fördern die weiteren Fortschritte des Teams in der Entwicklungsphase. Von dieser Phase wird oft behauptet, dass darin „die wirkliche Arbeit erledigt wird“, doch eigentlich liegt der Schwerpunkt auf der Erstellung der Produktlösung.

2.3.1 Ziele

- Alle Produkteigenschaften sind implementiert, wenn auch noch nicht vollständig optimiert.
- Das Produkt hat Betatests bestanden, und es liegt eine aktuelle Fehlerliste vor, wobei die Fehler nicht unbedingt korrigiert sein müssen.
- Teammitglieder und Kunden stimmen darin überein, dass die Produkteigenschaften der Zielvorstellung sowie dem Anwendungsentwurf entsprechen und erfolgreich implementiert wurden.
- Die Materialien zur Unterstützung der Benutzer sind festgelegt und stehen zum Testen und zur Stabilisierung bereit.

2.3.2 Verantwortungsbereiche

Rolle	Verantwortungsbereich
Produkt-Management	Verwaltung der Kundenerwartungen. Fortführung der Pläne für Weiterleitung von Produktinformationen an Kunden und Auftraggeber. Vorbereitung der Kunden auf Alpha- und Betaversionen
Programm-Management	Förderung der Kommunikation zwischen den einzelnen Projektteams. Koordination der bei untergeordneten Meilensteinen fälligen Zwischenlieferungen. Verantwortlich für Revision von Funktionsspezifikation, Projektplan, Projektzeitplan und Risikobewertung.
Entwicklung	Erstellung des notwendigen Codes, um alle Produkteigenschaften zu implementieren. Durchführung erster Funktionstests. Erstellung von Alpha- und Betaversionen.

Benutzerschulung	Erste Tests bzgl. Produkteignung. Koordination der Benutzer für Alpha- und Betatests. Erstellung erster Materialien zur Benutzerunterstützung wie Hilfeassistenten, Online-Schulungsmaterial und formale Benutzerschulungen. Erstellung von Benutzer- und Supportdokumentationen.
Test	Erstellung detaillierter Spezifikationen, Pläne, Einsatzszenarien, Daten und Skripts für Durchführung erster Funktionstests Prüfung von internen Zwischenlieferungen, Alpha- und Betaversionen. Fehler finden und verfolgen. Dokumentation des Testprozesses.
Logistik-Management	Erstellung von Supportmaterial und –dokumentationen bzgl. Logistik und Verteilung. Unterstützung der Alpha- und Betaproduktversionen.

Tabelle 7 Verantwortlichkeiten während der Entwicklungsphase

2.3.3 Ergebnisse

- Revidierte Funktionsspezifikation, die dem tatsächlich entwickelten Produkt entspricht. Sie umfasst jede genehmigte Änderung des Entwurfs. Zusätzlich können neue Funktionen in die Spezifikation aufgenommen werden, die für zukünftige Versionen vorgesehen sind.
- Revidierter Projektplan des speziell auf die Planung der Stabilisierungsphase eingeht. Er umfasst auch die Beschreibung der Endprodukte, die ausgeliefert werden sollen. Jede Änderung während der Entwicklungsphase wird eingearbeitet.
- Revidierter Projektzeitplan, besonders für die Zeitplanung der Stabilisierungsphase. Auch wird jede während der Entwicklungsphase aufgetretenen Änderung eingetragen.
- Revidiertes Risikobewertungsdokument, das sowohl bereits identifizierte als auch neu erkannte Risiken beschreibt. Zusätzlich Beschreibung der Risikomanagementpläne und deren Fortschritte bis zum aktuellen Zeitpunkt.
- Quelltext und ausführbare Dateien, die einer mit allen Funktionen ausgestatteten Version der aktuellen Anwendung entspricht und der erste Versuch des Teams ist, einen Freigabekandidaten zu erstellen.
- Materialien für Benutzer und Support wie Hilfsassistenten, Benutzerdokumentation und Schulungsunterlagen. Zusätzlich Beschreibung, wie die Anwendung ausgeliefert, installiert, konfiguriert, verwaltet und in Betrieb genommen wird.

- Testelemente wie Testpläne, Spezifikationen, Anwendungsfälle und Skripts, die alle Eigenschaften des neuen Produktes umfassen. Entwicklung automatisierter Testskripts.

2.3.4 Untergeordnete Meilensteine

Um die Projektziele erfolgreich abzuschließen, muss das Team untergeordnete Meilensteine für die Produktlieferungen definieren, indem es die Entwicklungsphase in verwaltbare Einheiten unterteilt. Die grundlegenden Meilensteine entsprechen normalerweise einem oder mehreren der folgenden Ergebnisse: interne Version, Alphaversion und Betaversion.

Mit Hilfe untergeordneter Meilensteine lässt sich der Fortschritt bei der Produkterstellung messen. Arbeiten mehrere Teams parallel an der Quellcodeerstellung, zwingen interne Versionen das Team zur Synchronisierung des Codes auf Produktebene. Damit werden die untergeordneten Meilensteine exakt abgegrenzt. Abhängig von der Projektgröße liegen die internen Versionen üblicherweise ein bis vier Monate auseinander.

Das Team sollte außerdem anstreben, während der Entwicklung Zeitpunkte zu definieren, ab denen Benutzeroberfläche und Datenbanken nicht mehr verändert werden. Endprodukte wie Dokumente für die Benutzerschulung sind sehr stark von der Benutzeroberfläche abhängig, und sie sind für Unterstützungs- und Schulungsmaterialien unabdingbar. Außerdem hängt die Implementierung von Funktionen in der frühen Entwicklungsphase häufig mit den Datenbankstrukturen zusammen. Je früher die Benutzeroberfläche und der Datenbankentwurf in der Entwicklungsphase festgeschrieben werden, umso weiniger Änderungen sind in der davon abhängigen Dokumentation und dem Code nötig.

Frühe interne Versionen konzentrieren sich auf riskante Entwurfsbereiche, um die Durchführbarkeit oder erforderliche Entwicklungsänderungen zu bestimmen. Dadurch werden Kosten und negative Auswirkungen von Entwurfsänderungen minimiert.

Setzt ein Team während des Entwicklungszyklus interne Versionen ein, dann muss unbedingt eine bekannte Zielvorstellung definiert werden, die später als Bemessungsgrundlage dient. Jede interne Version ist eine Erweiterung einer vorhergehenden Version. Sie repräsentiert einen Qualitätsanspruch, der erfüllt werden muss, bevor das Team den internen Meilenstein erreichen kann. Jede interne Version besitzt auch eine kurze Stabilisierungsphase, in der das Team die Produktqualität dem Qualitätsanspruch entsprechend verbessern kann.

Externe Produktversionen zeigen Kunden, Benutzern und anderen externen Auftraggebern die erfolgreiche Ausführung des Projektplans an. Abhängig von Projektgröße und –dauer ist eine Alphaversion nur selten mit allen Funktionen ausgestattet. Eine Alphaversion dient dem Projektteam zur Übung, ein Produkt vollständig freizugeben und sie dient auch als Testgrundlage für sonstige Ergebnisse wie etwa Unterstützungsmaterialien für Benutzer und Logistik.

Externe Betaversionen sind gewöhnlich mit nahezu allen Funktionen ausgestattet und weisen keine Leistungsoptimierungen auf.

2.3.5 Quelltext und ausführbare Dateien

Der Quelltext und die ausführbaren Dateien sind ein Ergebnis der Entwicklungsphase und repräsentieren eine vollständige und mit allen Funktionen ausgestattete Lieferung des Teams. Der von den Programmierern erstellte Code

muss den funktionalen Erfordernissen entsprechen. Zudem muss auf die Qualität des Codes geachtet werden.

Während des gesamten Projekts muss qualitativ hochwertiger Code entwickelt werden, auch unter Zeitdruck. Das Projekt kann langfristig sehr teuer werden, wenn Entwickler den Quelltext unsachgemäß erstellen, um durch eine schnellere Entwicklung der Anwendung kurzfristig Kosten einzusparen. Der Quellcode wird dadurch sehr leicht unwartbar.

Um Qualitätseinbußen durch Termindruck zu verhindern, sollte das Team Standards festlegen, die

- Entwickler zu einem methodischen und disziplinierten Ansatz bei der Programmierung zwingen, auch unter Zeitdruck.
- Entwickler ständig daran erinnern, dass die interne Qualität des Codes von Bedeutung ist.

Der Einsatz von Standards und einem Prüfprozess, der das Einhalten der Standards sicherstellt, wird sich darauf auswirken, wie Programmierer kodieren. Es muss allen klar sein, dass solche Standards keine optionalen Richtlinien, sondern ein Muss sind.

Programmierstandards konzentrieren sich traditionell auf die folgenden Bereiche:

- Benennung,
- Entwurf,
- Kommentierung,
- Programmierregeln und –verbote.

Der Schwerpunkt dieser Themen liegt gewöhnlich auf der Erstellung von *wiederverwendbarem Code*, d.h. Code, der von anderen Programmierern gelesen und eingesetzt werden kann.

Beispiele für solche Programmierrichtlinien finden sich in [1] und [2].

2.3.6 Testen von Elementen

Der Testprozess ist nicht auf die nachfolgende Stabilisierungsphase beschränkt, sondern auch ein integraler Bestandteil der Entwicklungsphase. Während der Planungsphase wurde bereits ein grundlegender Testplan festgelegt und mit der Ausarbeitung einer detaillierten Testspezifikation begonnen, die festlegt, wie das Team einzelne Funktionen testen soll. Die Testspezifikation wird mit dem Ende der Entwicklungsphase abgeschlossen, da die Funktionsmenge ab diesem Zeitpunkt nicht mehr erweitert wird.

Der Übergang zwischen der Entwicklungsphase und der Stabilisierungsphase wird vom Übergang von Funktionstests zu Eignungstests charakterisiert.

Funktionstests:

- *Komponententests* konzentrieren sich darauf, ob Eigenschaften vorhanden und funktionsfähig sind.
- *Check-in-Tests* sind schnelle, automatisierte Tests, die von den Entwicklern ausgeführt werden, bevor sie Code in ein Quellcodeverwaltungssystem einchecken.

- *Tests zur Verifizierung der Erstellung*, die nach der täglichen Erstellung ausgeführt werden, um zu überprüfen, ob das Produkt erfolgreich erstellt wurde.
- *Regressionstests* sind automatisierte Tests, die nach der täglichen Erstellung und deren Verifizierung ausgeführt werden, um zu überprüfen, dass die Codequalität nicht nachgelassen hat, d.h. bereits realisierte Funktionalität immer noch fehlerfrei ist.

Haben Entwickler sich das Ziel gesetzt, Fehler vor den Testern zu finden, führen sie Funktionstests aus!

Eignungstests sind:

- *Konfigurationstests* bestätigen, dass ein Produkt auf der Zielhardware und –software ausgeführt werden kann.
- *Kompatibilitätstests* bestätigen die Verträglichkeit mit anderen Programmen und unter Umständen mit älteren Versionen des Programms.
- *Belastungstests* belasten ein Produkt bis zu seinen Grenzen. Dazu gehören die Speichermangel im Arbeitsspeicher und auf der Festplatte, umfangreicher Netzwerkverkehr und eine hohe Benutzeranzahl.
- *Leistungstests* bewerten den gesamten Systemdurchsatz (Transaktionen pro Sekunde) und die Reaktionszeit (z.B. 95% aller Anfragen werden innerhalb einer Sekunde beantwortet).
- *Tests der Dokumentation und der Hilfedateien* konzentrieren sich auf falsche Inhalte und Abweichungen vom Produkt.

Eignungstests werden vom Testteam durchgeführt.

2.3.7 Fehlerüberwachung

Ein *Fehler* ist alles, was vor der Freigabe der Anwendung adressiert und behoben werden muss. Die weit verbreitete Meinung, dass Fehler immer Defekte sind, stimmt nicht. Defekte bilden eine Fehlerkategorie, doch Fehler umfassen mehr. Ein Fehler ist jeder Problempunkt, der sich aus der Verwendung der Anwendung ergibt, z.B.:

- Forderungen von Verbesserungen,
- Vorschläge, die bis zur Freigabe nicht mehr berücksichtigt werden können,
- Probleme durch Benutzervorlieben,
- Unvermeidbare Entwurfskonsequenzen,
- Defekte

Fehlerüberwachungsprozesse umfassen Aspekte wie Fehlerberichte, Prioritäten, Anweisungen, Behebung und Abschluss. Fehler müssen verfolgt und verwaltet werden, damit das Projektteam während der Stabilisierungsphase die notwendigen Entscheidungen treffen kann.

Vor Beginn des Fehlerüberwachungsprozesses muss eine Fehlersammlung, üblicherweise eine Datenbank, eingerichtet werden. Neue Fehler werden durch Eingabe in die Datenbank veröffentlicht. Jeder Tester gibt die Fehler ein, die er selbst gefunden hat. Neue Fehler werden als aktive Fehler gekennzeichnet.

Der Leiter des Entwicklungsteams weist den Fehlern Prioritäten zu und verteilt sie an bestimmte Entwickler, die sich um die Korrektur kümmern. Die Entwickler

korrigieren alle Fehler, die eine entwicklungsbasierte Behebung erfordern und testet diese. Ist der aktive Fehler korrigiert, ändert sich dessen Status. Danach prüft ein Tester die Qualität der Korrektur und stellt sicher, dass sie keine neuen Fehler nach sich zieht. Tritt ein neuer Fehler auf, dann trägt ihn der Tester in die Fehlerdatenbank ein und damit wird der gesamte Zyklus erneut gestartet. Wurde der ursprüngliche Fehler nicht erfolgreich korrigiert, dann reaktiviert der Leiter des Entwicklungsteams den Fehler.

Das Team (aber nicht ein einzelner Entwickler) kann immer entscheiden, den Fehler nicht zu korrigieren, sondern ihn stattdessen zu dokumentieren. Auf diese Weise entscheidet das Team, die Fehlerkorrektur zu verschieben und diese Arbeit nach der Produktfreigabe zu erledigen.

Die Fehlerklassifizierung bietet eine Möglichkeit, Prioritäten und Risiken zu erkennen. Die Klassifizierung zeigt zwei wichtige Aspekte auf: die Dringlichkeit, die sich auf die Auswirkungen des Fehlers auf die gesamte Anwendung bezieht und die Priorität, welche die Auswirkungen des Fehlers auf die Stabilität der Anwendung bewertet. Fehler müssen nicht nur gemeldet, sondern auch klassifiziert werden.

Eine typische Klassifizierung der Dringlichkeit sieht folgendermaßen aus:

- *Dringlichkeit 1: Systemausfall* Systemabstürze, Systemblockierungen, das System arbeitet sonst wie nicht mehr oder Datenverluste treten auf.
- *Dringlichkeit 2: Großes Problem* Schwer wiegender Defekt der Softwarefunktion, der nicht unbedingt zum Systemabsturz oder zu Datenverlusten führt.
- *Dringlichkeit 3: Kleines Problem* Defekt in der Funktion der Software, der gewöhnlich nicht Daten- oder Arbeitsverluste nach sich zieht.
- *Dringlichkeit 4: Trivial* Kleinere kosmetische Probleme, die von den meisten Benutzern unbemerkt bleiben.

Eine typische Klassifizierung der Prioritäten sieht folgendermaßen aus:

- *Priorität 1:* Die Anwendung kann nicht ausgeliefert werden und das Team erreicht den nächsten Meilenstein wahrscheinlich nicht.
- *Priorität 2:* Die Anwendung kann nicht ausgeliefert werden, doch das Team erreicht den nächsten Meilenstein wahrscheinlich dennoch.
- *Priorität 3:* Die Anwendung kann ausgeliefert werden und das Team erreicht den nächsten Meilenstein. Die Behebung dieser Fehler werden tendenziell nur dann korrigiert, wenn am Ende des Projektes genügend Zeit und die Behebung nicht riskant ist.
- *Priorität 4:* Forderungen von Verbesserungen und Fehler, die es nicht wert sind, korrigiert zu werden.

Ein Fehler wird dann abgeschlossen, wenn ein Tester geprüft hat, dass die Behebung des Fehlers nicht ein anderes Problem erzeugt und der Fehler höchstwahrscheinlich nicht mehr auftritt.

Fehler werden normalerweise wie folgt aufgelöst:

- *Korrigiert:* Der Entwickler hat den Fehler korrigiert, die Korrektur getestet, den Code geprüft, der Korrektur eine Produktfreigabenummer zugewiesen und den Fehler wieder an den Tester zurückgegeben, der ihn gemeldet hat.

- *Duplikat:* Der Fehler ist ein Duplikat eines anderen Fehlers, der schon in der Fehlerdatenbank eingetragen ist. Der doppelt eingetragene Fehler wird aufgelöst, geschlossen und mit dem Originalfehler verknüpft.
- *Verschoben:* Der Fehler wird nicht in der aktuellen Version behoben, aber wahrscheinlich in einer Folgeversion. Diese Auszeichnung wird eingesetzt, wenn das Team Vorteile darin sieht, den Fehler zu beheben, allerdings keine Zeit oder Ressourcen besitzt.
- *Gemäß des Entwurfs:* Das gemeldete Verhalten ist beabsichtigt und in der Spezifikation beschrieben.
- *Nicht reproduzierbar:* Der Entwickler kann den Fehler nicht verifizieren.
- *Nicht korrigieren:* Der Fehler wird in der aktuellen Version nicht behoben, weil das Team sich entschieden hat, dass sich die Mühe nicht lohnt oder weil das Management oder der Kunde dem Fehler keine Bedeutung zumisst.

2.4 Phase 4: Stabilisierung

Der Stabilisierungsprozess wird, wie die anderen Projektphasen auch, vom gesamten Projektteam abgeschlossen. Jedes Teammitglied besitzt eine eigene Verantwortlichkeit beim Erreichen des Primärziels, der Produktauslieferung.

Im Gegensatz zu den anderen Phasen des Entwicklungsprozesses wird diese Phase nicht durch Aktionsschritte charakterisiert, sondern durch die Zwischenversionen. Das Team kann mehrere externe Produktversionen ausliefern, während an der endgültigen Version gearbeitet wird.

Ein wichtiger untergeordneter Meilenstein, der dem Team anzeigt, dass es sehr nahe an der Produktfreigabe ist, wird allgemein Zero-Bug-Release (ZBR) genannt. ZBR ist die erste Zwischenversion des Produktes, in der alle aktiven Fehler in irgendeiner Form behoben sind, ob nun korrigiert, verschoben oder als unwichtig betrachtet.

Ist das Team der Ansicht, dass das Produkt potenziell für die endgültige Freigabe bereit ist, erstellt es als nächsten untergeordneten Meilenstein den Freigabekandidaten.

Mit der Auslieferung einer Zwischenversion müssen extensive Tests beginnen, um zu bestimmen, ob das Produkt fertig ist. Ist das der Fall, klassifiziert das Team diese Version als das endgültige Produkt. Die endgültige Produktversion ist der Freigabekandidat, bei dem sich alle Auftraggeber, Teammitglieder und Kunden darüber einig sind, dass diese Version ausgeliefert werden soll.

Die Stabilisierungsphase wird auch durch die Fertigstellung aller Elemente für die Produktunterstützung geprägt. Dazu gehören Dokumente zur Benutzerunterstützung, Installations- und Konfigurationsunterlagen und Schulungsprogramme für Mitarbeiter der Logistik und des Supports. Diese Unterlagen sind ebenfalls zu testen und freizugeben.

2.4.1 Ziele

- Produktauslieferung und -abnahme

2.4.2 Verantwortungsbereiche

Rolle	Verantwortungsbereich
Produkt-Management	Koordination der Zwischenproduktversionen mit dem Kunden. Planung der Produkteinführung.
Programm-Management	Verwaltung der Zwischenproduktversionen. Pflege des Projektzeitplans. Betreuung der Abnahme des fertigen Produktes durch den Kunden. Vorbereitung der Logistik und Supportgruppen auf die Produktfreigabe.

Entwicklung	Suchen, melden und korrigieren von Fehlern. Gewährleistung, dass Produktintegration vollständig ist und Tests erfolgreich bestanden werden.
Benutzerschulung	Fertigstellen der Materialien zur Benutzerunterstützung. Bestimmung und Koordination von Schulungspersonal. Koordination der Benutzerschulungen für Kunden, Endbenutzer, Logistik und Support.
Test	Ausführung der Testpläne. Suchen, melden und klassifizieren von Anwendungsfehlern. Verifizierung und Abmeldung von Fehlern.
Logistik-Management	Installation, Verteilung und Unterstützung der Zwischenproduktversionen. Planung der Auslieferung des fertigen Produkts.

Tabelle 8 Teamrollen und Verantwortlichkeiten in der Stabilisierungsphase

2.4.3 Ergebnisse

- *Endgültige Produktversion*, die Quelltext und ausführbare Dateien umfasst.
- *Produktfreigabedokumente* enthalten Informationen über die Freigabeversion und die letzten Änderungen.
- *Dokumente zur Benutzer- und Supportproduktivität* sind die endgültigen Version der Unterstützungsinformationen (Anwendungsdokumentation, Schulungsunterlagen und Support-Dokumentation).
- *Testergebnisse* sind Fehlerstatus und Datenbank
- *Projektarchiv* umfasst alle Produktinformationen, die während des Entwicklungsprozesses erstellt wurden, ob sie mit dem Produkt ausgeliefert werden oder nicht.
- *Projektdokumente* umfassen alle Stadien der Produktdokumentation einschließlich der Versionen zu den Hauptmeilensteinen.

3 Projektbewertung

Eine gründliche Projektbewertung analysiert das Projekt sorgfältig, um dessen Stärken und Schwächen in jeder Entwicklungsphase aufzudecken. Erst Bewertungen nach Projektabschluss setzten einen Prozess in Gang, die erfolgreichen Aktionen in den Entwicklungsprozess eines Unternehmens aufzunehmen, damit das Projektteam potentielle Risiken in zukünftigen Projekten erkennen und angehen kann.

Das Programm-Management ist für die Umsetzung des Bewertungsprozesses verantwortlich.

Vorteile

Aus Projektbewertungen ergeben sich Vorteile sowohl für die Personen als auch das Unternehmen:

- Der *Projektabschluss* hat insbesondere dann eine hohe Bedeutung, wenn sich das Projektteam bald auflöst. Der offizielle Abschluss unterstützt die Teammitglieder darin, sich einer neuen Aufgabe zuzuwenden und bietet die Möglichkeit, die Meinung eines jeden einzelnen dennoch aufzunehmen.
- Ein *Forum für die Teamkommunikation* bietet insbesondere den Teammitgliedern die Möglichkeit sich zu äußern, die bestimmten Teile des Projektes noch mit gemischten Gefühlen gegenüber stehen. Werden diese Gefühle nicht in einer kontrollierten Umgebung geäußert, können solche Emotionen vielleicht unter nicht wünschenswerten Umständen, am falschen Ort oder auf die falsche Weise zum Ausdruck kommen.
- Die *Festlegung von bewährten Verfahren* eine Basis für die Softwareentwicklung innerhalb des Unternehmens. Mit jeder Projektbewertung werden diese Standardverfahren hinterfragt, weiter entwickelt und überarbeitet.
- In *Feedback-Schleifen* werden alle Ergebnisse der Projektbewertung in einer Verfahrensbeschreibung zusammengefasst und als Eingabe für nachfolgende Projekte verwendet.

Zeitpunkt

Eine Projektbewertung findet statt, nachdem das Projekt abgeschlossen ist. Bei größeren Projekten ist dies auch schon mit dem Abschluss der einzelnen Phasen möglich. Der ideale Zeitpunkt einer Projektbewertung ist zwei (bei kleineren Projekten) bis fünf Wochen (bei größeren Projekten) nach Abschluss, damit das Team zum einen schon einen gewissen Abstand zum Projekt hat und zum anderen aber dieses noch detailliert genug im Gedächtnis sitzt.

Formalitäten

Bei einer Projektbewertung handelt es sich im Idealfall um eine formale, gut strukturierte Veranstaltung mit Zielen, einer Tagesordnung und einem definierten Ablauf. Die Teilnehmer sind auf die Veranstaltung vorbereitet.

Die Tagesordnung jeder Projektbewertung sollte die folgenden Fragen enthalten, die nicht in einer bestimmte Reihenfolge angesprochen werden müssen:

- Wie ist das Team mit dem Projektzeitplan zurecht gekommen?
- Wie effektiv wurden dem Projekt Ressourcen zugewiesen?

- Welche Stärken und Schwächen haben die Teammitglieder in jeder Projektphase entdeckt?
- Was lief beim Projekt schief?
- Gibt es Vorschläge der Teammitglieder zur Verbesserung des gesamten Prozesses?
- Haben die Teammitglieder Empfehlungen für zukünftige Projekte?

Das Programm-Management nimmt außerdem zusätzliche projektspezifische Fragen oder Themen in die Tagesordnung auf.

Länge

Obwohl es ein kann, dass Projektbewertungen häufige Besprechungen und tagelange Veranstaltungen beinhalten, ist eine einzige Bewertungsbesprechung mit einer Dauer von höchstens zwei Stunden ideal. Gegebenenfalls ist der Prozess in eine Folge mehrerer Besprechungen zu gliedern.

Rahmen

Eine aggressionsfreie Umgebung ist Voraussetzung dafür, die realen Projektthemen anzugehen. Nicht einzelne Teilnehmer am Projekt sollen im Mittelpunkt stehen, sondern der Entwicklungsprozess und die Aktionen des Teams als Ganzes. Niemals dürfen einzelne Personen oder Gruppen zum Ziel negativer Kritik werden.

Teilnehmer

Normalerweise sollte jeder an der Projektbewertung teilnehmen, der am Projekt beteiligt war, auch wenn es sich nur um einen kurzen Zeitraum handelt. Jeder soll die Möglichkeit zur Meinungsäußerung erhalten.

Wird die Bewertungsgruppe zu groß, wenn alle Teammitglieder eingeladen werden, dann sollten kleinere Projektbewertungen durchgeführt werden, in dem die Personen in Gruppen eingeteilt werden, die für einzelne Projektelemente verantwortlich sind. Repräsentanten jeder Projektbewertungsgruppen nehmen danach an einer letzten umfassenden Projektbewertungssitzung teil.

Moderieren

Der Moderator sollte Ordnung und Struktur in die Besprechung bringen und auch sicher stellen, dass sich die Teammitglieder auf die Themen konzentrieren und nicht einander angreifen. Er muss außerdem darauf achten, dass alle Punkte der Tagesordnung behandelt werden und dass sich jeder an der Diskussion beteiligen kann.

Es ist von Vorteil, wenn der Moderator nicht dem Projektteam angehört. Ist kein externer Moderator verfügbar, übernimmt das Programm-Management die Gesprächsleitung.

Protokollieren

Das Protokollieren von Diskussionsbeiträgen trägt dazu bei, dass die Teammitglieder ihre Gedanken prägnanter formulieren und bereits dokumentierte Themen vermeiden. Es ist von Vorteil, eine Person zum Protokollführer zu ernennen, die nicht direkt in das Projekt involviert war. Diese Person kann sich auf das Sammeln von Informationen konzentrieren und wird nicht unbedingt die Aussagen analysieren.

Die Projektbewertung wird dokumentiert, indem die Ergebnisse der Besprechung zusammengefasst werden mit Zeitplan, Namen der Betroffenen, der Liste der Empfehlungen und den Protokollen.

Nachdem das Bewertungsdokument vollständig ist, muss es allen Teammitgliedern vor der Veröffentlichung zur Prüfung offen stehen, damit noch Änderungen angemeldet werden können.

Am Beginn eines jeden neuen Projektes sollten die Empfehlungen von vorherigen Projekten zu einer Art „Selbstprüfung“ des neuen Projektes begutachtet werden. Zu verschiedenen Planungszeitpunkten während eines neuen Projektes sollte die Liste der Empfehlungen nachgeprüft und bewertet werden.

Wem, alles in allem, die vier Phasen des Entwicklungsprozessmodells und anschließend auch noch eine Projektbewertung jedoch zu aufwendig erscheinen, der kann auch wie bisher mit dem bewährten Sechs-Phasen-Prozess weitermachen:

Phase 1: Enthusiasmus

Phase 2: Desillusionierung

Phase 3: Panik

Phase 4: Suche nach den Schuldigen

Phase 5: Bestrafung der Unschuldigen

Phase 6: Lob und Ehren für die Unbeteiligten

4 Anhang

4.1 Mehrschichtenanwendungen

Viele Anwendungen arbeiten heute mit einer *Zwei-Schichten-Architektur* oder *Client-Server-Architektur*. Die Verarbeitung und Präsentation der Daten erfolgt auf dem Client, die Daten selbst sind auf einem zentral verwalteten Server abgelegt. Die Clients sind, oft nur für die Ausführungsdauer einer Anwendung, direkt mit den Servern verbunden, deren Daten sie benötigen.

Client-Server Architekturen sind sehr gut für kontrollierbare Umgebungen geeignet, in denen die Zahl der Benutzer überschaubar und verwaltbar ist und die Ressourcen entsprechend zugewiesen werden können. Wenn die Anzahl der Benutzer jedoch unbekannt ist oder einen gewissen Rahmen übersteigt, bricht die Client-Server-Architektur zusammen. Da jeder Client direkt mit den Datenservern verbunden ist, bleibt die Skalierbarkeit durch die Anzahl der verfügbaren Datenverbindungen beschränkt.

Eine kleine Verbesserung stellt das Verschieben von Teilen der Datenverarbeitungs- oder Geschäftslogik an den Datenserver dar, z.B. durch Verwendung von „Stored Procedures“ in der Datenbank. Skalierbarkeit und Wiederverwendbarkeit sind jedoch auch hier beschränkt.

Die Skalierbarkeit und Wiederverwendbarkeit können durch das Einbringen einer dritten Schicht in die Architektur entscheidend verbessert werden. Man spricht von 3-tier oder n-tier Anwendungen, in denen Benutzer-, Geschäfts- und Datenschichten logisch voneinander getrennt werden. Diese drei Schichten erfüllen folgende Aufgaben:

- *Benutzerschicht:* Die Benutzerschicht präsentiert dem Benutzer Daten und ermöglicht ggf. deren Bearbeitung. Benutzeroberflächen können systemspezifisch oder Web-basiert sein.
- *Geschäftsschicht:* Die Geschäftsschicht dient zum Umsetzen der Geschäfts- und Datenregeln. Die Benutzerschicht greift auf die Dienste der Geschäftsschicht zu.

  Geschäftsregeln können Geschäftsalgorithmen, Geschäftsrichtlinien etc. sein. Sie werden normalerweise in eigenen Codemodulen implementiert, die zentral gespeichert werden, sodass mehrere Anwendungen darauf zugreifen können.
- *Datenschicht:* Die Geschäftsschicht hat keine Informationen darüber, wo die Daten gespeichert werden, die sie bearbeitet. Sie verlässt sich dabei auf die Datenzugriffsdienste, welche die Daten abrufen und wieder zurück geben.

  Auch die Datenzugriffsdienste werden in eigenen Codemodulen implementiert, in denen auch Informationen über die zu Grunde liegenden Daten erfasst sind. Ändert sich der Standort oder das Format des Datenspeichers, müssen lediglich die Datenzugriffsdienste aktualisiert werden. Jedes Datenzugriffsmodul ist für die Integrität einer Datengruppe verantwortlich.

Die Begriffe *mehrschichtig* und *n-tier* deuten nicht auf separate Computer hin. Die n-tier Anwendungsarchitektur unterstützt skalierbare Anwendungen. Ressourcen, wie z.B. Datenbankverbindungen, müssen gemeinsam genutzt werden, wenn eine hohe Skalierbarkeit von Anwendungen erreicht werden soll.

Ressourcen können eingespart werden, wenn jede Client-Anwendung nicht direkt auf einen Datenserver zugreift, sondern mit einem Geschäftsdienst kommuniziert. Eine Instanz eines Geschäftsdienstes kann mehrere Clients versorgen, dadurch werden weniger Ressourcen verbraucht und die Skalierbarkeit verbessert.

Da Geschäftsdienste die Daten nicht direkt verwalten, können diese Dienste repliziert und dadurch noch mehr Clients unterstützt werden. Dienste können oft unabhängig von bestimmten Client-Anwendungen entworfen und implementiert werden und ermöglichen so Flexibilität und die Möglichkeit des mehrfachen Einsatzes in vielen Anwendungen. Darüber hinaus können allgemeine Funktionalitäten mühelos an geänderte Geschäftsanforderungen angepasst werden, ohne dabei die Client-Anwendungen, die mit diesen Funktionalitäten arbeiten, zu beeinträchtigen. Die Kosten für die Verwaltung und die Verteilung, die durch Anforderungsänderungen bedingt sind, werden so niedrig gehalten.

4.2 Begriffsdefinitionen

Begriffszuordnungen anderer Anwendungsentwicklungsmodelle

Begriff	MSF
Vorstudie (Grobanalyse)	„Phase 1: Zielvorstellung gewinnen“
Fachliche Feinanalyse	„Phase 2: Planung“ mit Schwerpunkt auf „Konzeptioneller Entwurf“, „Logischer Entwurf“ und „Funktionsspezifikation“
Technische Feinanalyse	„Phase 2: Planung“ mit Schwerpunkt auf „Physischer Entwurf“
Realisierung	„Phase 3: Entwicklung“
Test und Freigabe	„Phase 4: Stabilisierung“

Tabelle 9 Vergleich der Begriffe aus anderen Entwicklungsprozessmodellen

4.3 Ergänzende Unterlagen

[1] Ascherl, Christoph
„Styleguide für die C++ Entwicklung“
C++ Styleguide.doc

[2] Microsoft Cooperation
”Microsoft Foundation Class Library Development Guidelines”
MSDN Library