Kapitel 1
Software-Prozessmodelle

Ein Software-Prozessmodell ist ein *Modell* für die Entwicklung eines Software-Systems. Da Modellbildung immer auch Abstraktion beinhaltet, geht es nicht um die Darstellung des Ablaufs eines *bestimmten* Software-Entwicklungsprojekts, sondern einer ganzen *Klasse* von Projekten. Daraus ergibt sich, dass nicht jedes Prozessmodell für jede Klasse von Projekten geeignet ist. Es gibt grundsätzlich schwergewichtige und leichtgewichtige Prozessmodelle, wobei die agilen Modelle zu den leichtgewichtigen gehören.

1.1 Ein Prozessmodell – was ist das?

Ein Software-Prozessmodell ist ein *Modell* für den Ablauf der Entwicklung eines Software-Systems. Dabei geht es nicht um die Darstellung des Ablaufs eines *bestimmten* Software-Entwicklungsprojekts, sondern einer ganzen *Klasse* von Projekten. Dazu wird der Entwicklungsprozess üblicherweise in *Phasen* eingeteilt. Man unterscheidet typischerweise zwischen

- Planung des Prozesses,
- Spezifikation der Anforderungen an das Produkt,
- Design des Software-Produkts,
- Implementierung (Kodierung) und
- diversen Tests des Software-Produkts.

Im Rahmen eines Prozessmodells werden neben den *Aktivitäten* innerhalb dieser Phasen auch die *Rollen* und Qualifikationen der Mitarbeiter definiert, die gewisse Aktivitäten durchführen oder für sie verantwortlich sind. Außerdem werden die Dokumente und Unterlagen, zusammenfassend *Artefakte*[1] genannt, festgelegt, die im Rahmen des Entwicklungsprozesses erstellt werden müssen. Oftmals sind hierbei auch Vorgaben von Behörden zu beachten. Es gibt nicht *das* Prozessmodell für alle Typen von Software-Entwicklungsmodellen. Ausschlaggebend für die Wahl eines Prozessmodells ist der *Projekttyp*.

[1] Zum Begriff des Artefakts (engl. artifact) siehe (Jacobson et al. 1999).

E. Hanser, *Agile Prozesse: Von XP über Scrum bis MAP*, eXamen.press,
DOI 10.1007/978-3-642-12313-9_1, © Springer-Verlag Berlin Heidelberg 2010

1.1.1 Einteilung in Projekttypen

Eine frühe Einteilung in Projekttypen nimmt Barry Boehm bereits in den 1980er Jahren vor (Boehm 1981). Er unterscheidet zwischen „organic“, „semidetached“ und „embedded“ Projekten:

- *Organic Mode*-Projekte haben relativ kleine Entwicklungsteams, die Software in enger räumlicher Nähe entwickeln. Die Teammitglieder sind meist sehr erfahren, sowohl fachlich als auch methodisch. Meist wissen sie über das spätere Einsatzgebiet der Software recht gut Bescheid. Oft ist die Code-Größe der Software recht gering.[2]
- *Embedded Mode*-Projekte sind das Gegenteil zum oben besprochenen Projekttyp. Sie unterliegen starken „Zwangsbedingungen“, sind meist „reguliert“, d.h. von behördlichen Auflagen geprägt, oder sie unterliegen anderen Zwängen, z.B. den Unwägbarkeiten hardware-naher Projekte.[3] Die entwickelte Software muss hohen Anforderungen an die Zuverlässigkeit des Systems genügen, und meist sind nachträgliche Software-Anpassungen nahezu unmöglich. Das Entwicklungsteam ist meist groß. Dies hat auch damit zu tun, dass die Anforderungen an die Software detailliert beschrieben sind, was oftmals Platz für Parallelisierung der Entwicklungsarbeit schafft. Das Entwicklerteam besteht aus erfahrenen und unerfahrenen Mitgliedern.[4] Oft ist die Code-Größe der Software sehr hoch.
- *Semidetached Mode*-Projekte stehen *zwischen* den oben beschriebenen Projekttypen und bilden eine Mischung aus Organic und Embedded Mode. Die Software-Entwicklungsteams sind meist mittelgroß und bestehen aus erfahrenen und unerfahrenen Programmierern. Oft sind nicht alle Aspekte des Produkts bekannt. Die Code-Größe ist meist hoch.

1.1.2 Schwergewichtige und leichtgewichtige Prozessmodelle

Die vorgestellte Einteilung in Projekttypen ermöglicht auch eine entsprechende Einteilung der Software-Prozesse und ihrer Prozessmodelle in „schwergewichtige“ und „leichtgewichtige“ Modelle[5]:

- *Schwergewichtige* Prozessmodelle zeichnen sich durch eine sehr formale, dokumentengestützte Vorgehensweise aus. Jede Phase der Entwicklung wird ausführlich dokumentiert, der Prozess selbst ist in seinem Ablauf klar beschrieben und hält auch behördlicher Begutachtung stand. Der Einsatz schwergewichtiger Prozessmodelle ist sinnvoll, wenn eine Fehlfunktion des Software-Produkts eine

[2] Dies ist aber keine Bedingung für den Organic Mode.

[3] Die Entwicklung der Software in einem militärischen Projekt gehört sicherlich zu dieser Klasse.

[4] Dies ergibt sich alleine schon aufgrund der Größe des Teams.

[5] Der Begriff „leichtgewichtiger Prozess“ begegnete dem Autor zum ersten Mal in (Fowler u. Scott 2000).

Gefahr für Leib und Leben provozieren kann. Schwergewichtige Prozessmodelle gelten aber auch als unflexibel, insbesondere in Projekten mit wechselnden Anforderungen. Sie sind besonders geeignet für den Einsatz in *Embedded Mode*- und meist auch in *Semidetached Mode*-Projekten.

- *Leichtgewichtige* Prozessmodelle sind dagegen eher geeignet für Software-Projekte in kleinen Teams, wo die Anforderungen an das Produkt anfangs nur unvollständig bekannt sind. Ausführliche Spezifikationen machen damit wenig Sinn, da die Gefahr der Änderung zu groß ist. Gleichzeitig ist die Kommunikation im Team und mit dem Kunden aufgrund der kleinen Teamgröße sehr gut. Viele Informationen fließen informell, müssen also nicht explizit schriftlich fixiert werden. Die funktionierende Software steht im Vordergrund und nicht ihre Dokumentation. Leichtgewichtige Prozessmodelle sind besonders geeignet für den Einsatz in *Organic Mode*-, manchmal auch in *Semidetached-Mode*-Projekten.

Die in diesem Buch behandelten *agilen Prozessmodelle* gehören zur Gruppe der *leichtgewichtigen* Prozessmodelle.

1.1.3 Prozessmodell vs. Vorgehensmodell

Im deutschen Sprachraum wird anstelle des Begriffs Prozessmodell gerne das Wort *Vorgehensmodell* benutzt.[6] Grundsätzlich kann man diese Begriffe synonym verwenden. Es fällt allerdings auf, dass das in Deutschland dominierende Vorgehensmodell, das *V-Modell*, zumindest in der frühen Version *V-Modell 97* keine Aussagen über die Interna des eigentlichen Software-Entwicklungsprozesses machte.[7] Lediglich die Dokumentenschnittstelle zwischen Auftraggeber und Entwickler wurde umfangreich definiert.

Auch die gebräuchliche Rückübersetzung von Vorgehensmodell als „Life Cycle Model“ lässt den Schluss zu, dass es eine Differenzierung zum Begriff des Prozessmodells gibt. Der Autor wird deshalb in diesem Buch zwischen Prozessmodell und Vorgehensmodell unterscheiden. Immer wenn es um die Interna des *Software-Entwicklungsprozesses* geht, wird vom „Prozessmodell“ die Rede sein. Geht es mehr um den *Life Cycle* und den Blick „von außen“ auf die Entwicklung, wird der Begriff „Vorgehensmodell“ verwendet.

1.2 Das Phasenmodell

In den 1970er Jahren konnte man in der Software-Entwicklung noch davon ausgehen, dass die Anforderungen an das Software-Produkt am Anfang des Projekts

[6] Prozessmodell entspricht eher direkten Übersetzung des engl. *process model*.

[7] Dies wurde erst mit dem *V-Modell XT* nachgeholt. Zu empfehlen ist hierzu die Lektüre von (Höhn u. Höppner 2008). Zum V-Modell 97 siehe auch (Dröschel u. Wiemers 2000).

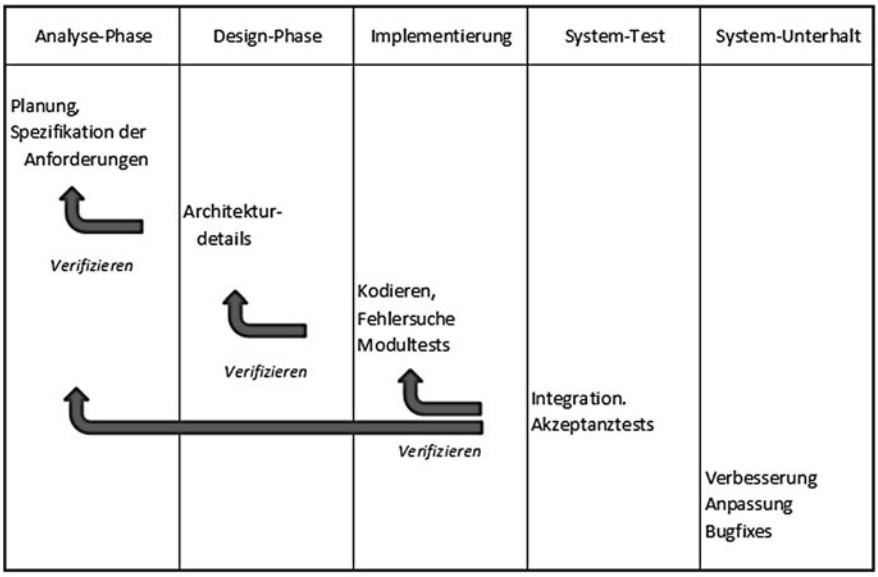

Abb. 1.1 Das Phasenmodell nach Royce

definiert wurden und sich bis zum Schluss nicht mehr änderten.[8] Royce entwickelte deshalb 1979 das Phasenmodell (Royce 1970).[9] In einer vereinfachten Form (Fairley 1985) zeigt es die fünf, zeitlich hintereinander folgenden Phasen des Software-Entwicklungsprozesses[10] (Abb. 1.1).

Aus Abb. 1.1 wird deutlich, warum das Phasenmodell auch „Wasserfallmodell" genannt wird. Die Phasen sind in Form von Kaskaden eines mehrstufigen Wasserfalls angeordnet. Die Rückpfeile sind *Verifizierungspfeile*. Nach der Kodierung eines Moduls wird beispielsweise verifiziert, ob das Modul mit den Vorgaben der Design-Phase, den Architekturdetails, übereinstimmt. So wird immer gegen die (zeitlich) vorhergehende Phase verifiziert. Erst bei den abschließenden Akzeptanztests wird nochmals gegen die Spezifikationen verifiziert.

1.3 Das Spiralmodell

In den 1980er Jahren wurde es immer dringlicher, Prozessmodelle zu entwickeln, welche es zulassen, eine Software in aufeinander aufbauenden Versionen zu

[8] War dies doch einmal der Fall, wurde das Projekt neu aufgelegt.

[9] Siehe hierzu auch (Boehm 1981) oder (Fairley 1985).

[10] Boehm kennt acht Phasen. Im Grunde detailliert er Abb. 1.1 nur weiter (Boehm 1981).

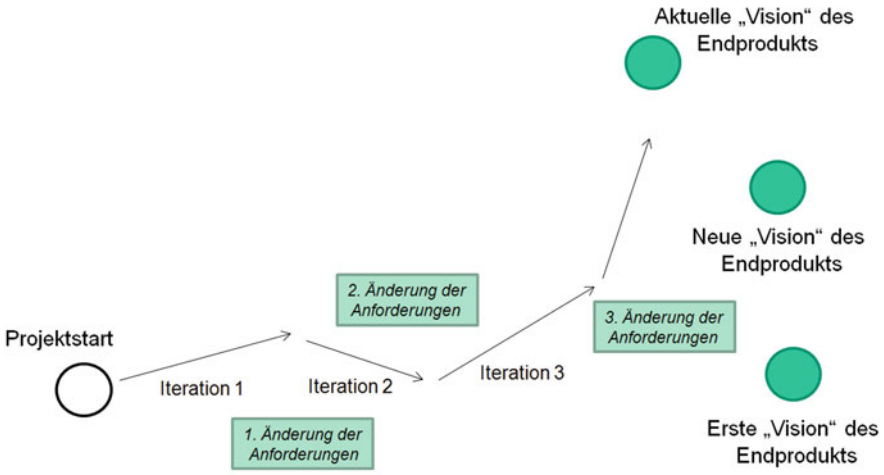

Abb. 1.2 Die Vision des Endprodukts als„Moving Target“

entwickeln. Das Phasenmodell war immer dann unzureichend, wenn Software in „Stücken“, sog. Iterationen, entwickelt werden musste. Dies war insbesondere dann der Fall, wenn die Anforderungen an ein Software-Produkt anfangs nicht bis ins Detail geklärt waren oder sich im Laufe des Projekts änderten („Moving Targets“, siehe Abb. 1.2).[11]

Barry Boehm schlug für solche Fälle Mitte der 1980er Jahre das sog. *Spiralmodell* vor.[12] Es beschreibt den sog. *iterativ-inkrementellen* Entwicklungsprozess von Software.

Der Entwicklungsprozess wird in zeitliche Abschnitte eingeteilt, in denen eine neue Funktionalität der Software implementiert wird, die sog. *Iterationen*. Für jede dieser Iterationen findet das Phasenmodell mit seinen Phasen Analyse bis Systemtest Anwendung. Das Phasenmodell wird also *pro Iteration* einmal durchlaufen. Am Ende jeder Phase steht das Inkrement (falls in der Iteration nicht lediglich Fehler oder Missverständnisse korrigiert wurden), die neue Funktionalität der Software. Ein oder mehrere Inkremente ergeben eine neue Release oder Version.

Das Spiralmodell ist genau genommen kein Prozessmodell, sondern ein „Meta Modell“. Auf dem Ansatz der iterativ-inkrementellen Software-Entwicklung setzen alle modernen Prozessmodelle auf.

[11] Bei der Entwicklung von Software für Maschinensteuerungen kam man auch weiterhin mit dem Phasenmodell aus.

[12] Siehe dazu (Boehm 1986) oder (Boehm 1988).

1.4 Moderne iterativ-inkrementelle Prozessmodelle

Zur Einteilung einiger aktueller Prozessmodelle soll die Klassifizierung in schwergewichtige und leichtgewichtige Prozessmodelle dienen. Als erstes seien zwei Vertreter der *schwergewichtigen Prozessmodelle* erwähnt:

1.4.1 V-Modell

Das V-Modell ist besonders in Deutschland und Österreich verbreitet und ist in der aktuellen Version *V-Modell XT* das Vorgehensmodell des Bundes (Höhn u. Höppner 2008). Es ist sehr stark im „regulierten" Umfeld verbreitet, besonders auch in der pharmazeutischen Industrie, dort aber meist in der historisch ältesten Version, dem V-Modell nach Boehm (Boehm 1979).

Kritiker bemängeln die starke „Dokumentenlastigkeit" des Modells, die die Projektkosten erhöht. Insbesondere in der pharmazeutischen Industrie werden deshalb Wege gesucht, kostengünstigere Vorgehensmodelle anzuwenden (siehe auch Kap. 7.7).

1.4.2 Unified Software Development Process (UP)

UP ist das Resultat der Entwicklung eines iterativ-inkrementellen Prozessmodells bei der Firma Rational in den USA[13] (Jacobson et al. 1999). Der UP besteht aus vier Phasen: *Inception* (Einstieg), *Elaboration* (Ausarbeitung), *Construction* (Konstruktion) und *Transition Phase* (Übergang) (Abb. 1.3).

Während in der Inception-Phase der Einstieg in das Projekt erfolgt, dient die Elaboration Phase zum Erstellen der Risikoanalyse, der Spezifikationen in Form von Use Cases und der Basisarchitektur des Systems. Außerdem wird die Construction Phase geplant, in der die eigentliche Systemerstellung stattfindet.

In der Construction Phase werden die Iterationen zur Systemerstellung durchgeführt. Pro Iteration finden die vom Phasenmodell her bekannten Phasen *Detailanforderungsanalyse*, *Detaildesign*, *Kodierung*, *(Modul-) Test* und *Integration* statt. Die Iteration endet mit einer Kundendemonstration.

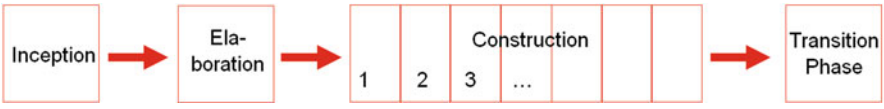

Abb. 1.3 Der Unified Process (UP)

[13] Rational wurde 2002 von IBM akquiriert.

Auch der UP gilt mit seinen vielen vorgeschriebenen Management- und Ingenieursartefakten als sehr dokumentenlastig (und damit als teuer).

1.4.3 Agile Prozessmodelle

Im Gegensatz zu den beiden bisher beschriebenen Prozessmodellen gelten die agile Modelle als *leichtgewichtig*. Nicht die Dokumentation steht im Vordergrund, sondern funktionierender Code. Trotzdem sind agile Projekte nicht anarchisch. Weder ist die Vorgehensweise im Projekt chaotisch, noch ist es verboten, zu dokumentieren. Lediglich die Gewichtung ist eine andere. Grundsätzlich dürfen nur Prozesse als *agil* bezeichnet werden, die dem *agilen Manifest* entsprechen. Dieses soll im nächsten Kapitel näher beleuchtet werden.

Literatur

Boehm B (1979) Guidelines for verifying and validating software requirements and design specifications. EURO IFIP 79, pp. 711–719, North Holland 1979

Boehm B (1981) Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall

Boehm B (1986) A spiral model of software development and enhancement. ACM SIGSOFT Software Engineering Notes, August 1986

Boehm B (1988) A spiral model of software development and enhancement. IEEE Computer, Vol. 21, Ausg. 5, Mai 1988, pp. 61–72

Dröschel W, Wiemers M (2000) Das V-Modell 97. München: Oldenbourg

Fairley R (1985) Software Engineering Concepts. New York, NY: McGraw-Hill

Fowler M, Scott K (2000) UML konzentriert. Bonn: Addison-Wesley

Höhn R, Höppner S (2008) Das V-Modell XT. New York, NY: Springer

Jacobson I, Booch G, Rumbaugh J (1999) The Unified Software Development Process. Reading, MA: Addison-Wesley

Royce W (1970) Managing the development of large software systems. Proceedings of IEEE WESCON, Vol. 26, pp. 1–9.