2.1 Vorgehensmodelle

Wie ich in *Kapitel 1* erläutert habe, handelt es sich bei einem Vorgehensmodell um eine vereinfachte Darstellung eines Softwareprozesses. Jedes Vorgehensmodell stellt einen Prozess aus einer bestimmten Sicht dar und liefert somit nur einen Teil der Informationen über diesen Prozess. Zum Beispiel zeigt ein Prozessaktivitätsmodell die Aktivitäten und ihre Abfolge, aber möglicherweise nicht die Rollen der Personen, die an diesen Aktivitäten beteiligt sind. In diesem Abschnitt führe ich einige sehr allgemeine Vorgehensmodelle (auch Prozessparadigmen genannt) ein und stelle sie aus architektonischer Sicht vor. Das heißt, wir sehen die Struktur des Prozesses, aber keine Details über die speziellen Abläufe.
Diese allgemeinen Modelle sind keine endgültigen Beschreibungen der Softwareprozesse. Vielmehr sind sie nützliche Abstraktionen des Prozesses, die benutzt werden können, um verschiedene Ansätze der Softwareentwicklung darzustellen. Sie können sich diese als Prozessrahmen vorstellen, die erweitert und angepasst werden können, um spezifischere Software-Engineering-Prozesse zu erzeugen.
Die Vorgehensmodelle, die ich in diesem Kapitel erläutere, sind:

1. *Das Wasserfallmodell:* Dieses Modell stellt die grundlegenden Prozessabläufe wie Spezifikation, Entwicklung, Validierung und Evolution als eigenständige Phasen des Prozesses dar, wie zum Beispiel Anforderungensspezifikation, Softwareentwurf, Implementierung, Tests usw.

2. *Inkrementelle Entwicklung:* Dieser Ansatz verknüpft die Aktivitäten der Spezifikation, der Entwicklung und der Validierung. Das System wird als eine Folge von Versionen (Inkremente) entwickelt, wobei jede Version neue Funktionalität zu der vorherigen hinzufügt.

3. *Wiederverwendungsorientiertes Software-Engineering:* Dieses Modell basiert auf der Existenz einer beträchtlichen Anzahl von wiederverwendbaren Komponenten. Der Systementwicklungsprozess beschäftigt sich mehr damit, diese Komponenten in ein System zu integrieren, als damit, neue Komponenten von Grund auf zu entwickeln.

Diese Modelle schließen sich nicht gegenseitig aus und werden häufig zusammen verwendet, vor allem bei der Entwicklung großer Systeme. Für große Systeme ist es sinnvoll, einige der besten Eigenschaften des Wasserfallmodells und des Modells der inkrementellen Entwicklung zu kombinieren. Man benötigt Informationen über die wesentlichen Systemanforderungen, um eine geeignete Softwarearchitektur zu entwerfen. Hier kann man nicht inkrementell vorgehen. Doch Subsysteme innerhalb eines größeren Systems könnten mit unterschiedlichen Ansätzen entwickelt werden. Teile des Systems, die bereits gut verstanden sind, können mithilfe eines wasserfallbasierten Prozesses spezifiziert und entworfen werden. Die Teile des Systems jedoch, die wie die Benutzerschnittstelle im Voraus schwierig zu spezifizieren sind, sollten immer unter Benutzung des inkrementellen Ansatzes entwickelt werden.

2.1.1 Das Wasserfallmodell

Das erste veröffentlichte Modell für die Softwareentwicklung wurde von allgemeineren Entwicklungsprozessen abgeleitet (Royce, 1970). Dies wird in ▶ Abbildung 2.1 veranschaulicht. Wegen der Kaskade von einer Phase zur nächsten wird dieses Modell

Wasserfallmodell oder der **Softwarelebenszyklus** genannt. Das Wasserfallmodell ist ein Beispiel eines plangesteuerten Prozesses – im Prinzip muss man alle Prozessaktivitäten inhaltlich und zeitlich planen, bevor man anfangen kann, an ihnen zu arbeiten.

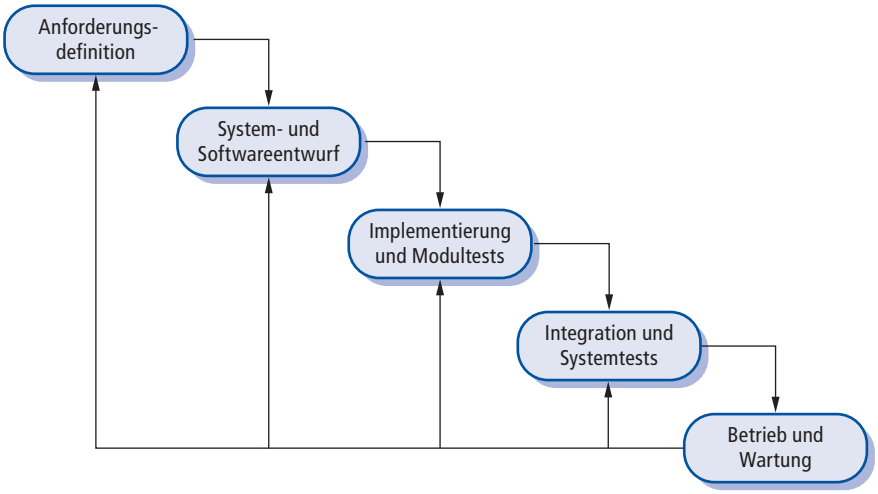

Abbildung 2.1: Das Wasserfallmodell.

Die wichtigen Phasen des Wasserfallmodells spiegeln direkt die grundlegenden Entwicklungsaktivitäten wider:

1. *Analyse und Definition der Anforderungen:* Die Dienstleistungen, Einschränkungen und Ziele des Systems werden in Zusammenarbeit mit den Systembenutzern aufgestellt. Dann werden sie detaillierter definiert und dienen so als Systemspezifikationen.

2. *System- und Softwareentwurf:* Der Systementwurfsprozess weist die Anforderungen entweder Hard- oder Softwaresystemen zu, indem eine übergeordnete Systemarchitektur festgelegt wird. Beim Softwareentwurf geht es um das Erkennen und Beschreiben der grundlegenden abstrakten Softwaresysteme und ihrer Beziehungen zueinander.

3. *Implementierung und Modultests:* In dieser Phase wird der Softwareentwurf durch eine Menge von Programmen oder Programmeinheiten umgesetzt. Das Testen der Module stellt sicher, dass jede Einheit ihre Spezifikation erfüllt.

4. *Integration und Systemtest:* Die einzelnen Programmeinheiten oder Programme werden integriert und als Ganzes getestet um sicherzustellen, dass die Softwareanforderungen erfüllt werden. Nach den Tests wird das Softwaresystem an den Kunden ausgeliefert.

5. *Betrieb und Wartung*: Normalerweise (aber nicht zwingend) ist dies die längste Phase innerhalb des Lebenszyklus. Das System wird installiert und zum Gebrauch freigegeben. Zur Wartung gehören das Korrigieren von Fehlern, die in den früheren Phasen nicht entdeckt wurden, die Verbesserung der Implementierung von Systemeinheiten und die Verbesserung des Systems, falls neue Anforderungen aufgedeckt werden.

Im Prinzip gehen aus jeder Phase ein oder mehrere Dokumente hervor, die genehmigt oder abgenommen werden. Die nächste Phase sollte nicht beginnen, bevor nicht die vorherige abgeschlossen wurde. In der Praxis überlappen sich die Phasen und tauschen Informationen untereinander aus. So werden während des Entwurfs Probleme mit den Anforderungen entdeckt. Während des Programmierens fallen Fehler im Entwurf auf usw. Der Softwareprozess ist kein einfacher linearer Prozess, sondern umfasst Rückkopplungen von einer Phase zur vorherigen. Die in der jeweiligen Phase erzeugten Dokumente müssen dann eventuell abgewandelt werden, um die vorgenommenen Änderungen zu reflektieren.

Wegen der hohen Kosten bei der Erstellung und Abnahme von Dokumenten können die Iterationen von Entwicklungsaktivitäten teuer sein und erhebliche Überarbeitungen verlangen. Daher wird normalerweise nach einer kleinen Anzahl von Wiederholungen ein Teil der Entwicklung, zum Beispiel die Spezifikation, eingefroren und eine spätere Entwicklungsstufe angefangen. Probleme werden dabei zurückgestellt, ignoriert oder im Programm umgangen. Dieses verfrühte Einfrieren von Anforderungen kann allerdings bedeuten, dass das System nicht das tun wird, was der Benutzer will. Es kann auch zu einem schlecht strukturierten System führen, weil Probleme im Entwurf durch Tricks in der Implementierung umgangen werden.

Während der letzten Phase des Lebenszyklus (Betrieb und Wartung) wird die Software in Betrieb genommen. Dabei werden Fehler und Lücken in den ursprünglichen Anforderungen entdeckt. Es tauchen Programm- und Entwurfsfehler auf und der Bedarf an neuer Funktionalität wird festgestellt. Das System muss sich also weiterentwickeln, um nützlich zu bleiben. Das Durchführen dieser Veränderungen (die Softwarewartung) kann zur Folge haben, dass einige oder alle der vorherigen Prozessphasen wiederholt werden müssen.

Das Wasserfallmodell ist konsistent zu anderen Systementwicklungsmodellen und erzeugt in jeder Phase Dokumentationen. Dadurch bleibt der Prozess durchschaubar, sodass Manager den Fortschritt gemäß Entwicklungsplan überwachen können. Das Hauptproblem ist jedoch die starre Aufteilung des Projekts in verschiedene Phasen. Zu einem frühen Zeitpunkt müssen Verbindlichkeiten eingegangen werden, was es schwer macht, auf neue Anforderungen des Kunden zu reagieren.

Im Prinzip sollte das Wasserfallmodell nur Verwendung finden, wenn die Anforderungen gut durchdacht sind und es eher unwahrscheinlich ist, dass es während der Systementwicklung zu gravierenden Änderungen kommt. Das Wasserfallmodell spiegelt jedoch die Art von Prozessen wider, die auch in anderen Ingenieurprojekten verwendet wird. Da es einfacher ist, ein gebräuchliches Managementmodell für das gesamte Projekt zu benutzen, sind noch immer Softwareprozesse, die auf dem Wasserfallmodell basieren, weitverbreitet.

Eine wichtige Variante des Wasserfallmodells ist die formale Systementwicklung, wobei ein mathematisches Modell einer Systemspezifikation erzeugt wird. Dieses Modell wird dann mithilfe von mathematischen Transformationen, die konsistenzerhaltend sind, zu ausführbarem Code verfeinert. Unter der Voraussetzung, dass Ihre mathematischen Transformationen korrekt sind, können Sie also überzeugende Argumente dafür vorbringen, dass ein so erzeugtes Programm konsistent mit seiner Spezifikation ist.

Formale Entwicklungsprozesse wie solche, die auf der B-Methode (Schneider, 2001; Wordsworth, 1996) basieren, sind besonders für die Entwicklung von Systemen mit strengen Betriebssicherheits-, Zuverlässigkeits- und Informationssicherheitsanforde-

rungen geeignet. Der formale Ansatz erleichtert es den Entwicklern den Nachweis zur Betriebs- und Informationssicherheit zu erstellen. Dies zeigt dem Kunden oder den Zertifikationsstellen, dass das System tatsächlich die Anforderungen an die Betriebs- oder Informationssicherheit erfüllt.
Prozesse, die auf formalen Transformationen basieren, werden im Allgemeinen nur in der Entwicklung von sicherheitskritischen Systemen eingesetzt. Sie erfordern besondere Fachkenntnisse. Für den Großteil von Systemen bietet dieser Prozess im Vergleich mit anderen Ansätzen zur Systementwicklung keine nennenswerten Vorteile in Bezug auf Kosten oder Qualität.

Cleanroom-Software-Engineering

Link

Ein Beispiel für einen formalen Entwicklungsprozess, der ursprünglich von IBM entwickelt wurde, ist der **Cleanroom-Prozess**. Im Cleanroom-Prozess wird jedes Softwareinkrement formal spezifiziert und diese Spezifikation wird in eine Implementierung umgewandelt. Die Korrektheit der Software wird durch einen formalen Ansatz bewiesen. In diesem Prozess gibt es keine Modultests und die Systemtests konzentrieren sich auf die Zuverlässigkeit des Systems.

Das Ziel des Cleanroom-Prozesses ist die Null-Fehler-Software, sodass ausgelieferte Systeme einen hohen Zuverlässigkeitsgrad haben.

http://www.SoftwareEngineering-9.com/Web/Cleanroom

2.1.2 Inkrementelle Entwicklung

Die inkrementelle Entwicklung basiert darauf, eine Anfangsimplementierung zu entwickeln, die Benutzer zu Kommentaren und Hinweisen zu dieser Implementierung aufzufordern und sie über mehrere Versionen hinweg zu verbessern, bis ein angemessenes System entstanden ist (▶ Abbildung 2.2). Die Spezifikation, die Entwicklung und die Validierung werden nicht als separate Abläufe betrachtet, sondern werden gleichzeitig ausgeführt, wobei sie untereinander Rückmeldungen zügig austauschen.

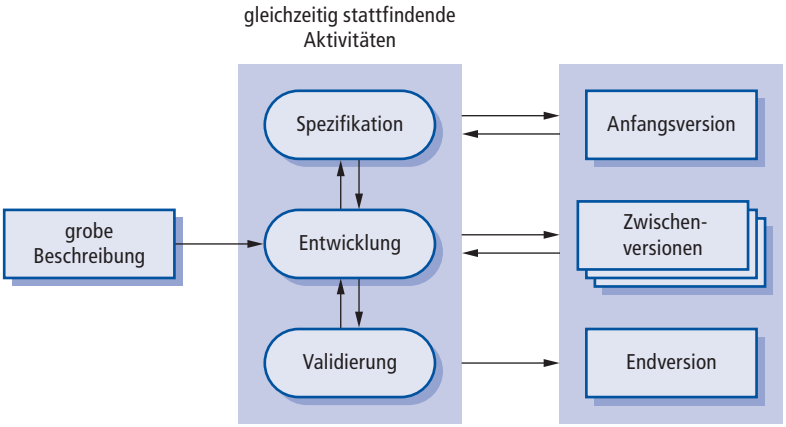

Abbildung 2.2: Inkrementelle Entwicklung.

Inkrementelle Softwareentwicklung ist ein fundamentaler Teil des agilen Ansatzes; sie ist besser als ein Wasserfallansatz für die meisten Geschäftsfälle, E-Commerce und individuellen Systeme geeignet. Inkrementelle Entwicklung bildet die Art und Weise ab, wie wir Probleme lösen. Wir erarbeiten selten eine vollständige Problemlösung im Voraus, sondern bewegen uns schrittweise in Richtung einer Lösung und gehen einen Schritt zurück, wenn wir feststellen, dass wir einen Fehler gemacht haben. Beim inkrementellen Entwickeln der Software ist es billiger und einfacher Änderungen einzuarbeiten als bei schon fertiger Software.

Jedes Inkrement oder jede Version des Systems besitzt einen Teil der Funktionalität, die vom Kunden gebraucht wird. In der Regel enthalten die frühen Systeminkremente die wichtigsten oder am dringendsten benötigten Funktionen. Dies bedeutet, dass der Kunde das System zu einem relativ frühen Zustand in der Entwicklung evaluieren kann um festzustellen, ob es den Anforderungen entspricht. Falls dies nicht der Fall ist, dann muss nur das aktuelle Inkrement geändert werden und es müssen eventuell neue Funktionen für spätere Inkremente definiert werden.

Inkrementelle Entwicklung hat gegenüber dem Wasserfallmodell drei wesentliche Vorteile:

1. Die Kosten für die Anpassung an sich ändernde Kundenanforderungen werden reduziert. Der Umfang der wiederholt durchzuführenden Analyse und Dokumentation ist geringer als beim Wasserfallmodell.

2. Es ist einfacher, Rückmeldungen der Kunden zu bereits fertiggestellten Teilen der Entwicklungsarbeit zu bekommen. Sie können sich bei Softwaredemonstrationen dazu äußern und sehen, wie viel implementiert wurde. Den Fortschritt anhand von Softwareentwurfsdokumenten zu beurteilen, fällt Kunden dagegen schwer.

3. Eine schnellere Auslieferung und Installation von verwendungsfähiger Software an den Kunden ist selbst dann möglich, wenn noch nicht die gesamte Funktionalität enthalten ist. Die Kunden können die Software früher verwenden und daraus Nutzen ziehen, als es mit einem Wasserfallmodell möglich wäre.

Inkrementelle Entwicklung in der einen oder anderen Form ist heute die am häufigsten eingesetzte Vorgehensweise für die Entwicklung von Anwendungssystemen. Dieser Ansatz kann entweder plangesteuert, agil oder – am häufigsten jedoch – eine Mischung dieser Methoden sein. In einem plangesteuerten Ansatz werden die Systeminkremente im Voraus bestimmt. Bei einem agilen Vorgehen werden zwar die frühen Inkremente ermittelt, die Entwicklung der späteren Inkremente hängt jedoch vom Projektfortschritt und von den Prioritäten des Kunden ab.

Die inkrementelle Entwicklung hat allerdings aus Managementsicht zwei Schwachpunkte:

1. Der Prozess ist nicht sichtbar: Manager brauchen in regelmäßigen Abständen Zwischenversionen, an denen sie den Fortschritt messen können. Wenn Systeme schnell entwickelt werden, ist es nicht kosteneffektiv, jede Version zu dokumentieren.

2. Die Systemstruktur wird tendenziell schwächer, wenn neue Inkremente hinzugefügt werden. Solange nicht Zeit und Geld in die Überarbeitung zur Verbesserung der Software investiert werden, besteht die Tendenz, dass stetige Veränderungen die Struktur der Software beeinträchtigen. Die Integration von weiteren Softwareänderungen wird zunehmend schwerer und teurer.

Bei großen, komplexen Systemen mit einer langen Lebensdauer werden die Probleme der inkrementellen Entwicklung schnell akut, wenn verschiedene Teams verschiedene Teile des Systems entwickeln. Große Systeme benötigen einen stabilen Rahmen oder Architektur und die Verantwortlichkeiten von unterschiedlichen Teams, die an Teilen des Systems arbeiten, müssen bezüglich dieser Architektur klar festgelegt werden. Dies muss im Voraus geplant statt inkrementell entwickelt werden.
Man kann ein System inkrementell entwickeln und es den Kunden zum Kommentieren vorführen, ohne es wirklich auszuliefern und in der Umgebung des Kunden einzurichten. Inkrementelle Auslieferung und Einrichtung bedeuten, dass die Software in realen betrieblichen Prozessen verwendet wird. Dies ist nicht immer möglich, da das Ausprobieren der neuen Software die normalen Geschäftsabläufe stören kann. Ich behandle die Vor- und Nachteile der inkrementellen Auslieferung in Abschnitt 2.3.2.

Probleme bei der inkrementellen Entwicklung

Link

Obwohl die inkrementelle Entwicklung viele Vorteile hat, so ist sie doch nicht ganz ohne Probleme. Der Hauptgrund für die Schwierigkeiten ist die Tatsache, dass große Organisationen bürokratische Handlungsabläufe besitzen, die sich im Laufe der Zeit entwickelt haben und die unter Umständen nicht mit einem informelleren iterativen oder agilen Prozess harmonieren.

Manchmal gibt es einen guten Grund für diese Handlungsabläufe – zum Beispiel um sicherzustellen, dass die Software gesetzliche Regelungen ordnungsgemäß umsetzt (z. B. die Vorschriften zur Rechnungslegung gemäß dem Sarbanes-Oxley Act in den Vereinigten Staaten). Diese Geschäftsabläufe zu verändern ist nicht immer möglich, sodass Konflikte zwischen den Prozessen unvermeidbar sind.

http://www.SoftwareEngineering-9.com/Web/IncrementalDev/

2.1.3 Wiederverwendungsorientiertes Software-Engineering

In einem Großteil aller Softwareprojekte wird Software wiederverwendet. Häufig geschieht dies informell, wenn die Mitarbeiter eines Projekts von einem Entwurf oder von Code wissen, der dem ähnelt, der gebraucht wird. Sie suchen ihn heraus, verändern ihn nach ihren Bedürfnissen und bauen ihn in ihr System ein.
Diese informelle Wiederverwendung findet unabhängig von dem eingesetzten Entwicklungsprozess statt. Im 21. Jahrhundert wurden jedoch Vorgehensmodelle der Softwareentwicklung immer populärer, die den Schwerpunkt auf die Wiederverwendung bereits vorhandener Software legten. Wiederverwendungsorientierte Ansätze beruhen auf einer großen Menge wiederverwendbarer Softwarekomponenten und auf einem Integrationsrahmen für die Zusammenstellung diese Komponenten. Manchmal handelt es sich bei diesen Komponenten um eigenständige käufliche Systeme (COTS-Systeme, *Commercial Off-The-Shelf System*), die benutzt werden können, um eine spezielle Funktion beizusteuern, wie zum Beispiel Textverarbeitung oder Tabellenkalkulation.
Ein allgemeines **Prozessmodell** für wiederverwendungsorientierte Entwicklung zeigt ▶ Abbildung 2.3. Obwohl die erste Phase der Anforderungsspezifikation und die Validierungsphase auch in anderen Softwareprozessen vorkommen, sind die Zwischenstufen in einem wiederverwendungsorientierten Prozess andere. Diese Stufen sind:

1. *Analyse der Komponenten:* Auf Basis der Anforderungsspezifikation wird nach Komponenten gesucht, mit denen diese Spezifikation implementiert werden kann. Meistens gibt es keine genaue Übereinstimmung, sodass die verwendeten Komponenten nur einen Teil der erforderlichen Funktionen zur Verfügung stellen.

2. *Anpassung der Anforderungen:* In dieser Phase werden die Anforderungen im Hinblick auf die gefundenen Komponenten analysiert. Dann werden sie an die verfügbaren Komponenten angepasst. Sollten Veränderungen nicht möglich sein, muss der Prozess wieder in die Phase der Komponentenanalyse eintreten, um nach alternativen Lösungen zu suchen.

3. *Systementwurf mit Wiederverwendung:* In dieser Phase wird der Rahmen für das System entworfen oder ein vorhandener Rahmen wiederverwendet. Die Entwickler betrachten die wiederverwendeten Komponenten und legen den Rahmen so an, dass er dazu passt. Wenn keine wiederverwendbaren Komponenten vorhanden sind, muss an dieser Stelle eventuell neue Software entworfen werden.

4. *Entwicklung und Integration:* Die Software, die nicht extern beschafft werden kann, wird entwickelt und zusammen mit den anderen Komponenten und den COTS-Systemen integriert. Die Systemintegration kann in diesem Modell auch Teil des Entwicklungsprozesses sein und muss nicht als eigenständiger Vorgang auftreten.

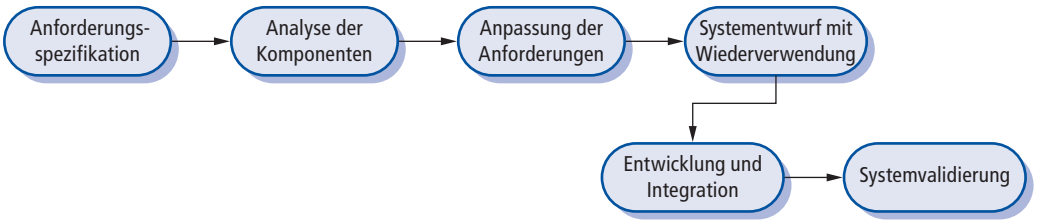

Abbildung 2.3: Wiederverwendungsorientiertes Software-Engineering.

Es gibt drei Arten von Softwarekomponenten, die in einem wiederverwendungsorientierten Prozess eingesetzt werden können:

1. Webdienste, die im Hinblick auf Servicestandards entwickelt werden und die für entfernte Aufrufe verfügbar sind;

2. Sammlungen von Objekten, die als Pakete entwickelt werden, um mit Komponenten-Framworks wie .NET oder J2EE integriert zu werden;

3. eigenständige Softwaresysteme, die für die Benutzung in einer bestimmten Umgebung konfiguriert wurden.

Wiederverwendungsorientiertes Software-Engineering hat den offensichtlichen Vorteil, dass es die Menge an zu entwickelnder Software und somit auch die Kosten und Risiken verringert. Außerdem wird die Software schneller geliefert. Kompromisse bei den Anforderungen sind jedoch unvermeidbar, und das kann zu einem System führen, das die wirklichen Bedürfnisse des Benutzers nicht erfüllt. Außerdem geht ein Teil der Kontrolle über die Weiterentwicklung des Systems verloren, da sich neue Versionen wiederverwendeter Komponenten der Kontrolle der Organisation entziehen, die sie benutzt.