2 Professionalisierung als Herausforderung

»Programmierung ist immer noch Flickwerk.
Es wird nur angeflanscht, erweitert, modifiziert
und am Ende werden Bugs quick-and-dirty beseitigt.«

Charles Symoni[1]

Die Disziplin der Softwareentwicklung ist mittlerweile über 40 Jahre alt. Dabei ist die Softwareindustrie gerade in den letzten 20 Jahren noch einmal stark gewachsen. Software spielt eine zentrale Rolle im täglichen Leben. Die Anzahl, die Größe und auch die Komplexität von Softwareanwendungen und ganzen Anwendungslandschaften sind dabei dramatisch gestiegen. Billionen von Dollars und Euro werden weltweit jährlich in die Softwareentwicklung investiert.

Das geht einher mit gewaltigen Kraftanstrengungen, Software »einfacher«, »besser« und »professioneller« zu entwickeln. Um das zu erreichen, gibt es gerade in den letzten 20 Jahren im Vergleich zu den Dekaden davor eine Flut neuer Methoden, Vorgehensweisen und Konzepten. Die Webseite *www.amazon.de* listet auf das Schlagwort »Softwareentwicklung« über 4.800 Bücher auf (Stand 01/2011). Dem Außenstehenden, aber auch dem Interessierten fällt es immer schwerer, sich in diesem Dschungel zu orientieren und die richtigen Entscheidungen für die eigenen Projekte zu treffen. Schließlich hat jeder den Anspruch, seine Software »gut« und »professionell« zu entwickeln.

Auf der anderen Seite stellt sich die Frage, wie heute in der Praxis wirklich Software entwickelt wird. Haben sich bestimmte Methoden durchgesetzt? Gibt es allgemeingültige Trends? Wo steht die Softwareentwicklung heute eigentlich? Trotz aller iterativen Konzepte, prototypischen Ansätze oder agilen Maximen geht es im Kern immer noch um die gleichen Aktivitäten und Aufgaben wie schon vor 20 Jahren.

Und: Ist die Softwareentwicklung wirklich professioneller und besser geworden? Haben die zahlreichen neuen Paradigmen und Methoden zu »besseren« Projekten, besseren Ergebnissen und besserer Software geführt? Können wir

1. Ex-Entwicklungschef von Microsoft auf einem Vortrag am MIT in Boston im September 2007.

heute von der Softwareentwicklung in der Breite als professionelle »Industrie« sprechen? Wenn nicht, woran liegt das? Was sind dann die Voraussetzungen für eine Professionalisierung der Softwareentwicklung?

In den folgenden Abschnitten versuchen wir, diese wichtigen Fragen zu beantworten. Wir wollen damit eine erste Orientierung über den Status quo der Softwareentwicklung geben. Wir wollen damit aber auch eine Grundlage für unser Kernthema der Produktivität schaffen. Für uns ist die Achse »Professionalität« und »Produktivität« eine zentrale Motivation des gesamten Buches. Eine professionelle Softwareentwicklung ist immer auch eine produktive Softwareentwicklung.

2.1 Wie wird heute Software entwickelt?

Zu den Grundlagen, Techniken und Methoden der Softwareentwicklung gibt es zahlreiche umfassende Arbeiten. Für einen umfänglichen Überblick über alle Bereiche der Softwareentwicklung verweisen wir den Leser auf den Klassiker von Ian Sommerville über Software Engineering [Sommerville 07] und das Buch von Jochen Ludewig und Horst Lichter [Ludewig & Lichter 10].

Die häufigste Organisationsform für Softwareentwicklung ist dabei heute das Projekt. Das ist unabhängig davon, ob ein Dienstleister für einen Kunden oder ob eine Entwicklungseinheit innerhalb eines Unternehmens Software entwickelt. Meist gilt das sogar für die Produktentwicklung, indem jede neue Version eines Softwareprodukts in Form eines Projekts durchgeführt wird. In diesem Sinne benutzen wir die Begriffe »Softwareentwicklung« und »Softwareentwicklungsprojekt« in der Regel als Synonyme.

Als Grundlage für das Thema unseres Buches wollen wir hier die folgenden Aspekte der Softwareentwicklung herausstellen und in ihrem Kern beschreiben:

- Aktivitäten
- Ergebnisse
- Methoden
- Reifegradmodelle

2.1.1 Aktivitäten der Softwareentwicklung

Abbildung 2–1 zeigt ein allgemeines Ablaufmodell für die Durchführung von Softwareentwicklungsprojekten. In jedem Entwicklungsprojekt sind fünf Basisaktivitäten durchzuführen:

- Anforderungsanalyse
- Design
- Implementierung
- Test
- Auslieferung

Die Basisaktivitäten werden in der Regel sequenziell durchlaufen. Aber auch eine teilweise Überlappung oder Parallelität ist möglich (zum Beispiel paralleles Implementieren und Testen). In einem Softwareentwicklungsprojekt können die Basisaktivitäten einmal oder mehrmals iterativ durchgeführt werden.

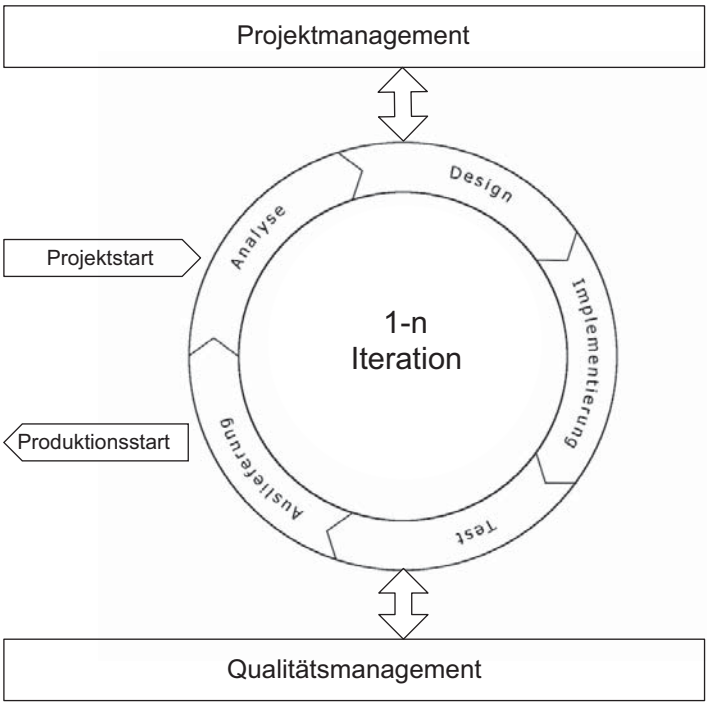

Abb. 2–1 *Die wichtigsten Aktivitäten der Softwareentwicklung*

Jede Basisaktivität produziert ein oder mehrere Ergebnisse. Wir sprechen von den »Ergebnistypen« einer Aktivität. Die Aktivität »Auslieferung« hat dabei die Besonderheit, unabhängig von der Anzahl der Iterationen der vorangehenden Aktivitäten entweder genau einmal oder ebenso mehrmals ausgeführt zu werden. Das hängt davon ab, ob die entwickelte Anwendung in mehreren sukzessiven Teilen (Inkrementen) ausgeliefert wird und in Produktion geht oder in einer einzigen Auslieferung.

Der konkrete Softwareentwicklungsprozess ergibt sich dann aus der sequenziellen oder parallelen Ausführung dieser Aktivitäten. Daher lässt sich der Kern jedes Entwicklungsprozesses wie folgt beschreiben [IEEE 90]: »Ein Softwareentwicklungsprozess ist ein Prozess, der Nutzeranforderungen in ein Softwareprodukt umsetzt. Der Prozess beinhaltet dabei die Übersetzung von Nutzeranforderungen in Softwareanforderungen, die Überführung der Softwareanforderungen in ein Design, die Implementierung des Designs in Code, das Testen des Codes und die Installation der Software für den praktischen operativen Einsatz.«

Daneben sind für die Softwareentwicklung drei zentrale Querschnittsaktivitäten herauszuheben, die primär steuernden Charakter haben:

- Projektmanagement
- Qualitätsmanagement
- Änderungs- und Konfigurationsmanagement

Auf der Grundlage dieser fünf Basisaktivitäten und drei Querschnittsaktivitäten kann man fast jedes individuelle Vorgehen, aber auch die meisten standardisierten Vorgehensmodelle (zum Beispiel V-Modell XT [Höhn & Höppner 09], Rational Unified Process (RUP) [Essigkrug & Mey 07], Scrum [Schwaber 04], Extreme Programming (XP) [Beck 00]) beschreiben. Die Terminologien für die verschiedenen Aktivitäten variieren natürlich in den Vorgehensmodellen und Entwicklungsmethoden. Insbesondere unterscheiden sich die konkreten Ausgestaltungen dieser Aktivitäten und ihre Ausführungsreihenfolge.

2.1.2 Ergebnisse der Softwareentwicklung

Softwareentwicklung ist ein produktiver Prozess, an dessen Ende ein nutzbares »Produkt« steht. Das Produkt und zentrale Ergebnis jeder Softwareentwicklung ist eine ausführbare Anwendung, die den Nutzer dieser Anwendung einen bestimmten Mehrwert liefert (Komfort, Zeit, Kosten). Im Laufe eines Projekts werden zusätzlich Dokumente und Zwischenergebnisse produziert, die für den Entstehungsprozess hilfreich waren, die aber kein Teil des Endergebnisses sind (siehe Tab. 2–1). Bei den Ergebnissen der Softwareentwicklung unterscheiden wir daher zwei Arten:

- **Endergebnisse oder Liefereinheiten**, die mit der Auslieferung »geliefert« werden und die für den täglichen Betrieb und den Nutzen der entwickelten Anwendung notwendig sind,
- **Zwischenergebnisse**, die im Projektverlauf entstanden sind, aber nicht ausgeliefert werden und für den Betrieb und Nutzen der Anwendung nicht unbedingt notwendig sind.

Basisaktivität	Typische Ergebnistypen
Anforderungsanalyse	■ Anforderungsliste ■ Anforderungsspezifikation ■ User Stories ■ Use Cases ■ UML-Modell ■ GUI-Prototyp ■ Fachkonzept
Design	■ Architekturdokument ■ Verfeinertes UML-Modell ■ DV-Konzept ■ Technischer Prototyp
Implementierung	■ Code ■ Systemdokumentation ■ Benutzerhandbuch
Test	■ Testkonzept ■ Testplan ■ Testfälle ■ Testprotokolle ■ Fehlerberichte ■ Abnahmeprotokoll
Auslieferung	■ Installierte Anwendung

Tab. 2–1 *Typische Ergebnisse der verschiedenen Basisaktivitäten*

Typische Endergebnisse und Liefereinheiten eines Entwicklungsprojekts sind:

- Ausführbare Anwendung (Code)
- Benutzerhandbuch
- Systemdokumentation
- Release Notes

Kann ein mit der Unified Modeling Language (UML) formuliertes Modell auch Liefereinheit und damit Ergebnis eines Projekts sein? Natürlich, wenn das zwischen dem Entwicklungsprojekt und dem Auftraggeber des Projekts so vereinbart ist, dann kann auch das UML-Design eine Liefereinheit und damit ein definiertes Endergebnis der Softwareentwicklung sein. Will ein Kunde etwa die Software, die ein Dienstleister für ihn entwickelt hat, selbst weiterentwickeln und warten, dann können solche Ergebnisse auch für den Kunden relevant sein. Entscheidend ist in diesem Sinne, welche Liefereinheiten mit dem Projektauftrag vereinbart sind. Diese Vereinbarung bestimmt, welche Endergebnisse ein Entwicklungsprojekt neben dem eigentlichen Code produziert und liefert.

2.1.3 Methoden der Softwareentwicklung

Begriffe wie »Vorgehensmodell«, »Entwicklungsmethode« oder »Vorgehensweise« werden für die Softwareentwicklung oft in einer ähnlichen Bedeutung verwendet. Dabei meint der Begriff des Vorgehensmodells ein umfassendes Ablaufmodell für den gesamten Entwicklungsprozess und dessen Aktivitäten (zum Beispiel Rational Unified Process, RUP). Allen Vorgehensmodellen ist gemeinsam, dass sie den Ablauf des Entwicklungsprozesses systematisch regeln und damit helfen, die Komplexität von Softwareentwicklungsprojekten zu beherrschen. Dabei werden vor allem die folgenden Vorgaben gemacht (siehe Abb. 2–2):

- Durchzuführende Aktivitäten und deren Reihenfolge
- Zu erstellende Ergebnisse und Teilergebnisse
- Verwendete Methoden und Werkzeuge
- Verantwortlichkeiten und Kompetenzen (Projektrollen)

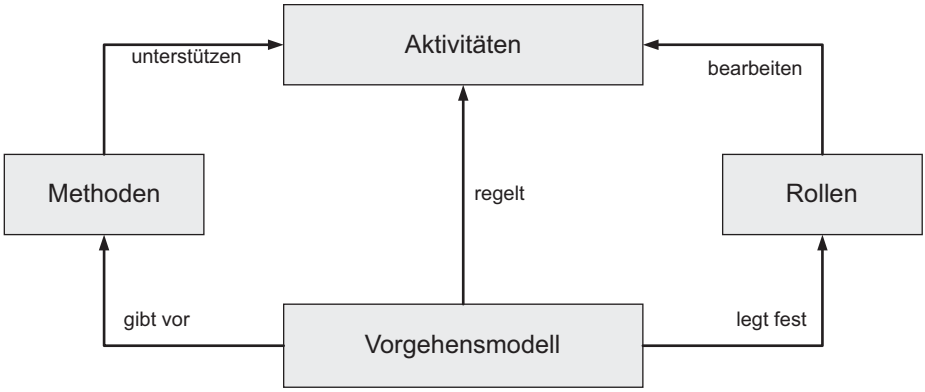

Abb. 2–2 *Zusammenhang Vorgehensmodell, Methode und Entwicklungsaktivität*

Der Begriff der Entwicklungsmethode betont dagegen mehr das Spezifische eines kleineren Schrittes oder einer einzelnen konkreten »Vorgehensweise« innerhalb einer Entwicklungsaktivität (zum Beispiel *Use Cases* zur Anforderungsanalyse, Aufwandsschätzung auf der Basis von *Function Points*). Dabei ist der Übergang von einzelnen Methoden und Sammlungen von Methoden zu Vorgehensmodellen fließend. So sind Vorgehensweisen wie Extreme Programming [Beck 00] oder Scrum [Schwaber 04] eher Methoden oder besser Sammlungen von Methoden als umfassende Vorgehensmodelle.

Bezüglich ihrer Philosophie unterscheidet man heute grob vier Kategorien von Vorgehensmodellen bzw. Entwicklungsmethoden (siehe auch [Fritzsche & Keil 07]):

- Wasserfallmodell
- Inkrementelle Entwicklung
- Iterative Entwicklung
- Agile Entwicklung

Im Kern unterscheiden sich die Entwicklungsphilosophien im Konzept der Iterationen, deren Anzahl, deren Längen und deren Ablauf (siehe Tab. 2–2). Dabei gibt es in der Praxis immer häufiger auch Mischformen. Das klassische Wasserfallmodell zum Beispiel tritt in Reinform kaum mehr auf.

Etwas vereinfacht kann man feststellen, dass heute vor allem zwei Typen von Vorgehensweisen dominieren, je nachdem, ob eher ein sequenzieller »langer« Wasserfall mit oder ohne Inkrementen oder eher das Konzept der kurzen Iterationen und der Agilität das Fundament für das Vorgehen bilden. Meist spricht man dann von »klassischen« oder »traditionellen« Vorgehensweisen gegenüber »iterativen« oder »agilen« Vorgehensweisen. Dabei waren die beiden Lager der Vertreter von agilen und von traditionellen Methoden vor ein paar Jahren noch im starken Wettstreit. Wie alles Neue brauchten auch die agilen Methoden ihre Zeit, um zum *Mainstream* zu werden und von (fast) allen akzeptiert zu werden. Heute haben sich die Schulen von agilen und traditionellen Methoden zum Teil bereits stark angenähert, sodass der Übergang zwischen den verschiedenen Entwicklungsphilosophien zunehmend fließend wird.

Agile und iterative Vorgehensweisen sind vor allem in dynamischen Projekten und Projektumfeldern gefragt. Das sind Projekte, in denen die Anforderungen nicht sehr stabil sind und während der Entwicklung mit vielen neuen und geänderten Anforderungen zu rechnen ist. Inwieweit das ohnehin eines der latenten Risiken in jedem Entwicklungsprojekt ist, sei hier dahingestellt. Agile Vorgehensweisen setzen Flexibilität in Bezug auf Änderungen (Anforderungen, Leistungsumfang, Projektziel) und damit das sogenannte *Moving Target* in das Zentrum ihrer Entwicklungsrhythmen.

Agilität ist dabei aber nicht nur eine Methode, Software zu entwickeln, sondern auch eine »Haltung«, verbunden mit einer bestimmten Projektkultur, die sich am agilen Manifest[2] orientiert. Werte wie Kundennutzen, Qualität und Teamgeist werden explizit in den Vordergrund gerückt. In eine ähnliche Richtung gehen Methoden, die sich dem sogenannten »Lean Management« [Poppendieck et al. 10] verschreiben.

2. Agiles Manifest siehe *http://agilemanifesto.org*.

Vorgehens-modell	Anzahl Iterationen	Typische Länge einer Iteration	Anzahl Anforderungs-analysen	Anzahl Auslieferungen
Wasserfall	1	1–3 Jahre	1	1
Inkrementell	n	3–6 Monate	n	n
Iterativ	nn	Mehrere Wochen	nn	1 bis nn
Agil	nnn	1–4 Wochen	nnn	1 bis nnn

Tab. 2–2 *Klassifikation verschiedener Vorgehensweisen*

Traditionelle Vorgehensweisen nach dem Wasserfallmodell und auch inkrementelle Ansätze setzen eine gewisse Stabilität bei der Planbarkeit und bei den Anforderungen voraus. Aus dieser Perspektive sind traditionelle Methoden eher gefragt für Entwicklungsprojekte in stabilen, wohlverstandenen Umfeldern (technisch und fachlich), wo man aus Erfahrung weiß, dass sich die Anforderungen nur geringfügig ändern und Architektur und Technik bereits mehrfach erfolgreich eingesetzt wurden. Beispiel ist die Weiterentwicklung einer bestehenden Anwendung, die bereits erfolgreich in Betrieb ist.

2.1.4 Die Bedeutung von CMMI und Reifegradmodellen

Zusätzlich zu konkreten Entwicklungsmethoden und umfassenden Vorgehensmodellen haben sich für die Softwareentwicklung sogenannte Reifegradmodelle etabliert. Die Grundidee von Reifegradmodellen ist es dabei, ein Rahmenwerk zur Verfügung zu stellen, an dem sich Unternehmen bei der Gestaltung der eigenen (Softwareentwicklungs-)Prozesse orientieren können. Das Rahmenwerk gibt dann im Wesentlichen Anforderungen an die Prozesse vor. Die konkrete Umsetzung mit konkreten Methoden wird vom Reifegradmodell nicht vorgegeben. Das ist der Unterschied zu den oben beschriebenen Vorgehensmodellen und Entwicklungsmethoden.

Reifegradmodelle haben dabei primär die Qualität von Prozessen und deren Verbesserung im Fokus. Damit unterscheiden diese Modelle »reife« von »unreifen« Organisationen. Der Reifegrad ist dann ein Indikator für die Zuverlässigkeit einer Organisation, ihre Entwicklungsprojekte immer wieder, also wiederholbar, erfolgreich durchführen zu können. Reifegradmodelle sind häufig mit einer Zertifizierung verknüpft. Wichtige Reifegradmodelle sind u. a.:

- CMMI
- ISO 9000/9001
- ITIL
- SPICE

Dabei ist das international bekannteste Reifegradmodell das *Capability Maturity Model Integration* (CMMI) des Software Engineering Institute (SEI) der Carnegie Mellon University [Chrissis, Konrad & Shrum 07]. CMMI ist nach unserer Einschätzung auch das im Umfeld betrieblicher Informationssysteme mit Abstand am weitesten verbreitete Reifegradmodell. Im CMMI sind dabei verschiedene Vorgängermodelle (CMM) integriert. CMMI kennt fünf Reifegrade einer Organisation, Softwareentwicklungsprojekte durchzuführen (siehe Tab. 2–3). Die Reifegrade unterscheiden sich im Grad der Kontrolle und Zuverlässigkeit der Entwicklungsprozesse und -projekte.

Level	Merkmale	Herausforderungen	Ergebnis
5 Optimizing	Kontinuierliche Verbesserung	Ressourcenintensiv, Kostenoptimierung	Produktivität und Qualität
4 Managed	Quantitativ, Messungen, Metriken	Kontinuierliche Verbesserung	
3 Defined	Qualitativ, standardisierte Prozesse	Qualitätsmessungen, Prozessmessungen	
2 Repeatable	Intuitiv, individuelle Prozesse	Best Practices, Prozesse von Einzelnen abhängig	
1 Initial	Ad hoc, chaotisch	Planung, Qualität, Projektmanagement	Risiko

Tab. 2–3 *Die fünf Stufen des CMMI-Reifegradmodells*

Stufe 1 beschreibt eine »Ad hoc«-Softwareorganisation, die keine oder sehr wenige Prozesse definiert hat. Es gibt keine Vorgaben zur Planung und Steuerung von Projekten. Organisationen, die Stufe 2 erreichen, haben zumindest grundlegende Prozesse und Methoden für ein kontrolliertes Projektmanagement. Tabelle 2–4 zeigt die sieben Prozesse, die für CMMI-Stufe 2 gefordert sind.

Requirements Management	REQM	Anforderungsmanagement
Project Planning	PP	Projektplanung
Project Monitoring and Controlling	PMC	Projektsteuerung
Supplier Agreement Management	SAM	Management von Zulieferungen
Measurement and Analysis	MA	Messung und Analyse
Process and Product QA	PPQA	Qualitätssicherung
Configuration Management	CM	Konfigurationsmanagement

Tab. 2–4 *Exemplarische Übersicht aller Prozesse der CMMI-Stufe 2*

Auf Stufe 3 ist ein organisationsweit gültiger Softwareentwicklungsprozess definiert und eingeführt, inklusive Projektmanagement und Qualitätssicherung. Dieser Prozess ist auch etabliert und funktioniert in allen Projekten.

Auf CMMI-Stufe 4 werden die Qualität der (Zwischen-)Ergebnisse und die Produktivität der Prozesse quantitativ gemessen. Metriken und Kennzahlen unterstützen die Projektsteuerung. Organisationen auf Stufe 5 haben darüber hinaus Methoden und Wege zur ständigen Prozessverbesserung etabliert.

Ein weitverbreiteter Irrtum liegt in der Ansicht, die Anwendung von CMMI führe notwendigerweise zu schwergewichtigen und teureren Prozessen, CMMI sei zu komplex und propagiere einen Wasserfallprozess. Es gibt inzwischen viele Beispiele, in denen konkrete agile Verfahren wie Scrum oder XP mit Erfolg auf eine bestimmte CMMI-Stufe abgebildet wurden [Glazer et al. 08]. CMMI ist eben nur ein *Framework*, das auch für agile Vorgehensweisen funktionieren kann.

Während CMMI vor allem in Nordamerika, Asien und auch Europa verbreitet ist, wird das Reifegradmodell ISO 9000/9001 vor allem in Europa eingesetzt. Es hat seinen Ursprung in der Fertigungsindustrie und ist im Vergleich zu CMMI viel allgemeiner formuliert. Im Kern ist es ein Regelwerk zur Qualitätssicherung innerhalb einer Organisation. ISO 9001 kennt nur zwei Stufen, d.h., ein Unternehmen erfüllt die Anforderungen oder es erfüllt sie nicht.

Während CMMI ein Rahmenwerk für die Entwicklung von Softwaresystemen ist, ist ITIL (*Information Technology Infrastructure Library*) ein Rahmenwerk für die Wartung und den Betrieb von Softwaresystemen. ITIL ist im Wesentlichen eine Sammlung von bewährten Verfahren aus dem IT-Servicemanagement.

Betrachtet man die verschiedenen Reifegradmodelle, angeführt von CMMI, stellt sich die Frage, inwieweit diese Modelle in der Praxis auch Nutzen stiften. Nutzen stiften in dem Sinne, dass Projekte von »reifen« Organisationen mit besserer Qualität, geringeren Kosten und höherer Produktivität durchgeführt werden. Dafür gibt es tatsächlich empirische Indizien, die wir in Teil IV aufgreifen werden.