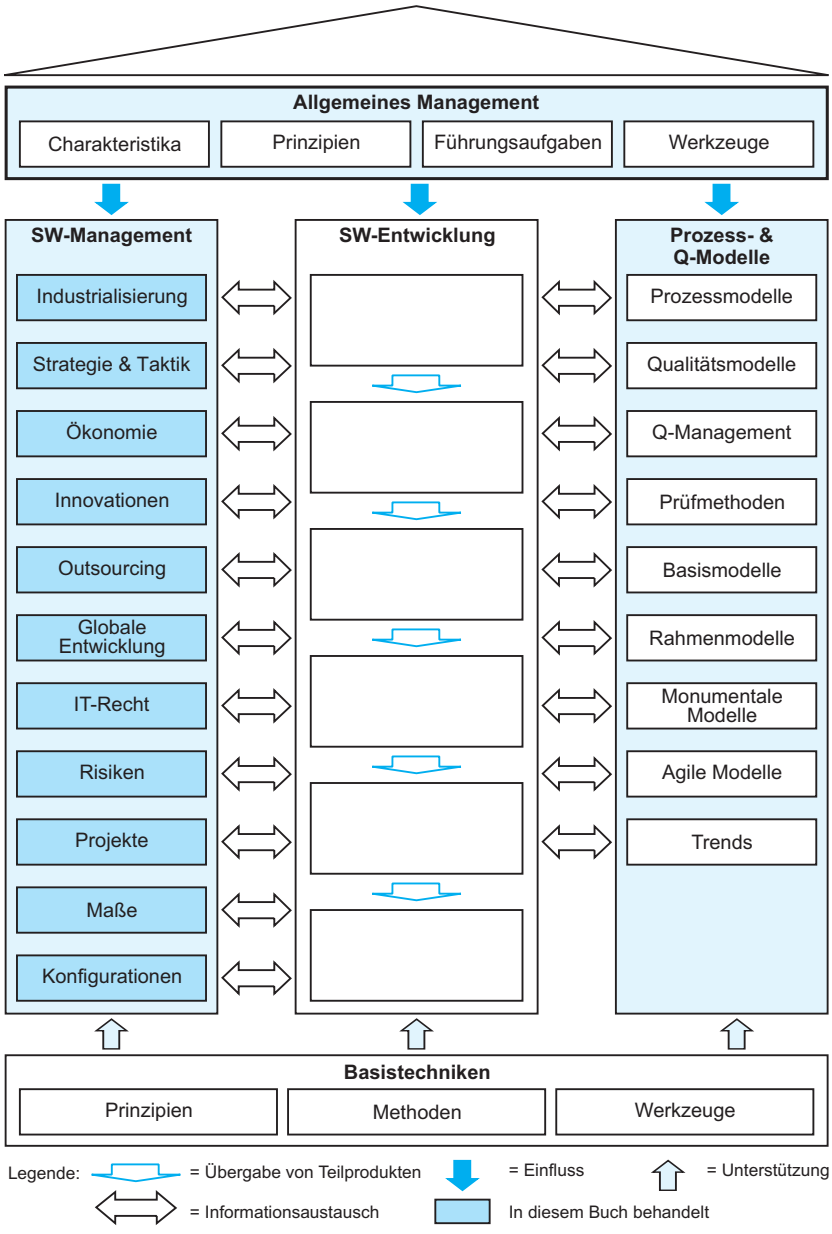
II Softwaremanagement



II II Softwaremanagement

Um einen Software-Bereich oder ein Software-Haus erfolgreich zu managen, müssen neben allgemeinen Management-Fähigkeiten (siehe »Allgemeines Management«, S. 1) die Sachaufgaben des Softwaremanagements beherrscht werden. Jeder Softwaremanager muss sich darüber im Klaren sein, dass das Management von Softwareentwicklungen eine Reihe von Besonderheiten und damit verbunden auch Schwierigkeiten aufweist:

- »Softwaremanagement – Was ist anders?«, S. 149

Die Entwicklung von Software-Strategien gehört zu den Kernkompetenzen eines Softwaremanagers:

- »Strategie und Taktik«, S. 161

Jeder Softwaremanager muss Software ökonomisch entwickeln. Er muss daher sehr genau wissen, welche Größen seine Ökonomie beeinflussen und wie er auf sie einwirken kann:

- »Softwareökonomie«, S. 189

Die hohe Innovationsgeschwindigkeit im Bereich der Softwaretechnik führt dazu, dass Softwaremanagement permanent mit der Einführung von Innovationen zu tun hat. Es gehört zu den wichtigsten Sachaufgaben zu wissen, wie man Innovationen einführt:

- »Einführung von Innovationen«, S. 213

Eine Möglichkeit, Kosten und Zeit zu sparen, besteht darin, Aufgaben der Softwareentwicklung an Lieferanten zu vergeben:

- »Outsourcing«, S. 237

Hat man sich entschlossen, Aufgaben »nach außen« zu vergeben, dann stellt sich die Frage an wen und wo:

- »Globale Softwareentwicklung«, S. 281

Bei jeder Softwareerstellung sind heute eine Vielzahl von rechtlichen Fragen zu berücksichtigen:

- »IT-Recht«, S. 313

Um Softwareprojekte erfolgreich durchzuführen, sind ständig die Risiken »im Auge« zu behalten:

- »Risiken managen«, S. 359

Zur Steuerung und Kontrolle von Softwareprojekten werden Maße benötigt:

- »Maße definieren, einführen und anwenden«, S. 377

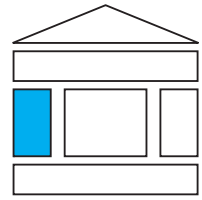
Alle diese Gesichtspunkte sind bei der Planung und Kontrolle von Softwareprojekten zu berücksichtigen:

- »Projekte planen und kontrollieren«, S. 393

Verschiedene Kunden erhalten oft unterschiedliche Softwarekonfigurationen. Während der Entwicklung und in der Einsatzphase treten Änderungen auf:

- »Konfigurationen und Änderungen managen«, S. 427

5 Softwaremanagement – Was ist anders?



Die Aussage *»Software is everywhere«* in [Charette 05, S. 37] kennzeichnet gut, dass Software etwas Besonderes ist. Ein durchschnittliches Unternehmen gibt 4 bis 5 Prozent seines Umsatzes für Informationstechniken aus. Unternehmen, die stark von der IT abhängig sind, wie Finanz- und Telekommunikationsunternehmen, geben mehr als 10 Prozent aus. Die IT-Ausgaben eines Unternehmens sind nach den Personalkosten die größten.

Die Boston Consulting Group schätzt, dass die Nachfrage nach industrieller Software um jährlich 10,5 Prozent steigt.

Fehler haben Auswirkungen

Die Kehrseite der Medaille: Führt Software zu Fehlern, dann kann sie Unternehmen sehr schnell in ihrer Existenz gefährden. *»IT failures can also stunt economic growth and quality of life«* [a. a. O., S. 39]. Will man wissen, was bei Softwareentwicklungen alles schief gehen kann, dann werden in der Regel die CHAOS-Reports der Standish Group (`http://www.standishgroup.com/`) zitiert. Nach den CHAOS-Reports führten Softwareprojekte zu folgenden durchschnittlichen Kostenüberschreitungen: 1994: 189 %, 1996: 142 %, 1998: 69 %, 2000: 45 % und 2002: 43 %. Andere Studien zeigten durchschnittliche Kostenüberschreitungen von 30 % auf. Eine Untersuchung des Beratungshauses Infora GmbH bei mehr als 300 Firmen mit einem Jahresumsatz von mindestens 100 Millionen Euro hat ergeben, dass nur 19 Prozent der Unternehmen mehr als 90 Prozent ihrer IT-Projekt vollständig und mit den gewünschten Ergebnissen abgeschlossen haben. 39 Prozent aller Firmen mussten zumindest jedes vierte Projekt als Flop verbuchen[1]. In [Charette 05, S. 38] ist eine *Software Hall of Shame* aufgeführt, die zeigt, welche großen Projekte von 1992 bis 2005 gescheitert sind. Der Autor kommt zu folgendem Schluss: *»If you define failure as the total abandonment of a project before or shortly after it is delivered, and if you accept a conservative failure rate of 5 percent, then billions of dollars are wasted each year on bad software.«*

Spannungsfeld 1

Als Softwaremanager steht man also zwischen den folgenden Extremen: Man kann mit seiner Mannschaft eine tolle Software entwickeln, die einem viel Lob und Gewinn einbringt. Man kann aber auch total scheitern und viel Geld »in den Sand setzen«.

[1] Quelle: Infora GmbH, zitiert aus der Computerwoche 28/2007, S. 13.

Abgesehen von reinen Softwarehäusern untersteht ein Softwaremanager in der Regel dem IT-Verantwortlichen oder dem Verantwortlichen für die Entwicklung – insbesondere bei eingebetteter Software. Im angelsächsischen Bereich unterscheidet man noch zwischen dem CIO *(Chief Information Officer)* und dem CTO *(Chief Technology Officer)*. Während der CIO für die IT-Planung, die IT-Technologieauswahl und den IT-Betrieb verantwortlich ist, ist der CTO der Leiter der technischen Entwicklung. Je nach Firmenstruktur gibt es einen CIO und/oder CTO. Im Rahmen der Globalisierung werden diese Begriffe auch zunehmend in Deutschland verwendet.

Zitat

»In den Boom-Zeiten der vergangenen Jahre waren CIO die einzigen, die etwas von der Technik verstanden. Was technisch machbar war, wurde gemacht, koste es, was es wolle. [...] Viel mehr als Technik konnten sie nicht, aber mehr als Technik verlange auch keiner von ihnen. Das katapultierte sie bis in die Vorstandsetagen. [...] Herabgestuft ins Mittelmanagement müssen sie mit zusammengestrichenen Budgets wirtschaften und Begriffe wie Return on Investment lernen. Und ihre Chefs stellen ganz andere Ansprüche an sie: „Der erfolgreiche CIO ist primär ein guter Businessmanager und erst sekundär ein guter Informatiker“« [Gillies 04, S. 95 f.] (siehe auch [Stannat, Petri 04]).

Spannungsfeld 2

Diese sich wandelnden Einstellungen zur IT wirken sich natürlich auch auf den Softwaremanager aus. Er steht im Spannungsfeld zwischen seinem CIO und seinen Mitarbeitern. Konzepte wie *Value-based Software Engineering, Evidence-based Software Engineering, Empirically-based Software Engineering* und *Total Cost of Ownership* sollten ihm vertraut sein, Soft Skills sollte er beherrschen.

das Einmalige

Software hat einmalige Eigenschaften. Dadurch bedingt können Erkenntnisse aus anderen Bereiche oft nicht oder nur eingeschränkt auf den Softwarebereich übertragen werden:

- »Das Besondere am Softwaremanagement«, S. 151

hin zur Industrie

Die Erstellung von Software entwickelt sich vom (Kunst-)Handwerk hin zu einer industriellen Produktionsweise, die gravierende Auswirkungen auf das Softwaremanagement hat:

- »Vom Kunsthandwerk zur Industrialisierung«, S. 152

welche Position?

Die Sachaufgaben eines Softwaremanagers hängen stark davon ab, ob er auf der Lieferanten- oder Auftraggeberseite tätig ist, ob er es mit Individual- oder Standardsoftware zu tun hat:

- »Das Umfeld eines Softwaremanagers«, S. 157

Eine Zusammenfassung beschließt dieses Kapitel:

- »Zusammenfassung«, S. 159

5.1 Das Besondere am Softwaremanagement

Besonderheiten

Softwaremanagement unterscheidet sich vom Management anderer Ingenieurbereiche aus folgenden Gründen:

- Das Produkt ist immateriell.
  Man sieht es nicht. Man kann es nicht berühren. Der Manager ist abhängig von der Dokumentation, um den Entwicklungsfortschritt zu überprüfen.
- Der Entwicklungsfortschritt ist objektiv *nicht* zu ermitteln.
  Es ist schwer festzustellen, ob ein Teilprodukt fertiggestellt ist oder nicht. Ein Softwareentwickler kann fast jederzeit behaupten, er sei mit einem Teilprodukt fertig, d. h. es existiert ein Dokument oder ein Quellprogramm. Um festzustellen, ob dieses Dokument oder Quellprogramm überhaupt verwendbar ist, ist eine genaue Untersuchung durch einen Experten erforderlich. Um die Qualität zu überprüfen, muss man fast denselben Aufwand betreiben wie bei der Entwicklung. Der Softwaremanager ist also voll auf die Aussagen seiner Mitarbeiter angewiesen. Es fehlen in der Regel einfache und billige Kontrollmittel (siehe auch: »Maße definieren, einführen und anwenden«, S. 377). Sogar gute Entwickler glauben oft, fertig zu sein und sind es nicht, da die Aufgabe nicht eindeutig definiert wurde.
  Natürlich gibt es auch genug Entwickler, die den wahren Stand ihrer Arbeit verdecken, um Zeit zu gewinnen. Bei Großprojekten wird die Verschleierung des Projektzustandes oft von den Managern der mittleren Ebene geschickt fortgesetzt, so dass es noch schwieriger wird, den wahren Zustand zu ermitteln [Sneed 87, S. 203].
- Eine Softwareentwicklung verläuft nicht-deterministisch.
  Neue Erkenntnisse während der Entwicklung haben Auswirkungen auf die bisherigen Ergebnisse. Deshalb ist ein bestimmtes Teilprodukt immer nur bedingt fertig. Diese entwicklungsinternen Zyklen müssen bei der Projektplanung mitberücksichtigt werden. Oft wird die Hälfte der Arbeit zur Überarbeitung bereits erstellter Teilprodukte benötigt [Sneed 87, S. 203 f.].
- Es gibt noch kein klares Verständnis vom Entwicklungsprozess (siehe »Prozess- und Qualitätsmodelle«, S. 445).
  Andere Ingenieurdisziplinen haben eine lange Historie. Die Entwicklungsstufen dort sind verstanden und vorhersagbar.
- Große Softwaresysteme tendieren dazu, einmalige Entwicklungen zu sein.
  Sie unterscheiden sich von früheren Entwicklungen. Gemachte Erfahrungen sind von begrenztem Wert, um vorherzusagen, wie das Management dieser Entwicklungsprozesse aussehen sollte.

- Unteilbarkeit der Arbeit.
  Um ein Softwareproblem zu lösen, muss sich der Entwickler intensiv mit dem Problem beschäftigen. Dadurch wird die Übertragung von Aufgaben an einen anderen Mitarbeiter teuer, da der neue Mitarbeiter die Einarbeitungszeit des alten Mitarbeiters wiederholen muss.
  Aufgaben sind also nicht voll austauschbar. Nicht jeder Mitarbeiter kann die Aufgaben eines anderen übernehmen, denn dafür sind die Aufgaben oft viel zu spezialisiert. Fällt ein Experte aus, dann kann man ihn nur durch einen vergleichbaren Experten ersetzen. In Netzplänen wird jedoch oft von der vollen Austauschbarkeit der Mitarbeiter ausgegangen [Sneed 87, S. 204 f.] (siehe auch »Projekte planen und kontrollieren«, S. 393).
- Die Softwaretechnik ist *keine* Naturwissenschaft.
  Als ein künstliches Produkt des menschlichen Erfindungsgeistes basiert Software *nicht* auf physikalischen Prinzipien und wird daher auch nicht durch diese begrenzt. Jedoch gibt es empirische Gesetze zu Qualität, Aufwand, Änderungsraten usw.
- Es liegt ein hoher Grad an Abstraktion vor bei gleichzeitig niedrigem Grad an Normierung [Sneed 87, 205].

Diese Unterschiede gegenüber Entwicklungen anderer Disziplinen sind für einen Außenstehenden nur schwer zu erkennen. Daher scheitern viele Manager aus anderen Branchen, wenn sie versuchen Softwareentwicklungen zu managen.

Viele Softwaremanager sind von Managementideen geprägt, die aus typischen Produktionsprozessen stammen. Wegen der Besonderheiten einer Softwareentwicklung ist *mehr* Management als bei Produktionsprozessen erforderlich, nicht weniger. Viele Softwareentwicklungen werden zu spät fertiggestellt, die Kosten und die Termine werden überschritten. Jede Entwicklung eines großen Softwaresystems ist ein neues und technisch innovatives Projekt. Viele Ingenieurprojekte, die ebenfalls innovativ sind, z. B. neue Transportsysteme, neue Brücken, neue Tunnel, haben ebenfalls Zeit- und Kostenprobleme.

5.2 Vom Kunsthandwerk zur Industrialisierung

Um die Sachaufgaben eines Softwaremanagers zu identifizieren, ist es wichtig zu wissen, welche Art von Softwareentwicklung zu managen ist. Die Softwareentwicklung durchläuft einen Weg vom Software-Kunsthandwerk hin zu einer industriellen Softwareerstellung.

In vielen Software-Häusern und Software-Abteilungen wird heute Software noch so erstellt, wie in einer Manufaktur oder einem Handwerksbetrieb. Manufaktur bedeutet: Vorindustrieller gewerblicher Großbetrieb mit Handarbeit. In [Janßen 05, S. 285] wird folgende Beobachtung wiedergegeben:

Zitat

»Dennoch haben sich die Entwicklungsabteilungen in Unternehmen oft nach dem Kunsthandwerkermodell organisiert: Für jede Anwendung, jede Anwendergruppe gab es eine mehr oder minder feste Gruppe von Mitarbeitern, die diese Anwendung ganzheitlich betreuten, neue Anforderungen festlegten, diese umsetzten, technische Weiterentwicklungen durchführten, usw.«

Als Nachteile dieser Organisationsform führt Janßen Folgendes auf:

1. Die Ressourcen sind einem Zweck fest zugeordnet und daher nur in geringem Maße beweglich.
2. An der einmal gewählten Technik wird festgehalten – der Wechsel auf eine Standardsoftware wird nicht in Betracht gezogen.
3. Der Generalist überwiegt: Jeder macht von jedem ein bisschen. Die Professionalität bleibt auf der Strecke (siehe auch: »Rollen und ihr Kompetenzprofil«, S. 98).
4. Man verlernt das Lernen, da man immer auf demselben Themenumfeld arbeitet. Wird eine Anwendung dann doch durch eine Standardsoftware ersetzt, dann bleibt nur noch ein Einsatz im Support übrig (siehe auch: »Personalentwicklung in der Softwaretechnik«, S. 109).

Dieses Entwicklungsmodell ist vielleicht effizient, wenn der Veränderungsdruck und die Veränderungsgeschwindigkeit gering sind [a. a. O., S. 285].

Zitat

»Jeder Softwareentwickler kann heute nur noch bei wenigen Technologien eine Wissenstiefe und Erfahrung sammeln, die ihn in die Lage versetzen, sie optimal anzuwenden. Je breiter das Einsatzgebiet eines Entwicklers ist (heute *Frontend*, morgen Geschäftslogik, übermorgen *Deployment*), desto durchschnittlicher kann nur das Resultat sein« [Westphal 05].

Die Software-Welt ist heute nicht mehr so wie oben dargestellt – abgesehen vielleicht von einigen Nischen. Aufgrund der globalen Wettbewerbssituation besteht ein Zwang, auch in der Softwareerstellung zur Industrialisierung zu kommen:

Definition

Software-Industrialisierung ist die Erstellung von Software mit industriellen Mitteln.

Was sind industrielle Mittel? Die Tab. 5.2-1 zeigt in Anlehnung an [Taubner 05, S. 293] und [Hochstein et al. 07] einen Vergleich zwischen klassischen Industrien und der Softwarebranche.

Klassische Industrien	Softwarebranche	Kommentar
Massenproduktion	Kopie der Software	Kopierkosten vernachlässigbar, aber Entw.-kosten auf hohe Stückzahlen verteilbar
Arbeitsteilung, Fließband	Zerlegung der Arbeit, Vorgehen	Prozessmodelle teilen die Aufgaben grobgranular auf, Festlegung von Rollen
Spezialisierung	zunehmende Spezialisierung	Teilaufgaben werden an Lieferanten vergeben, Festlegung von Rollen (siehe »Rollen und ihr Kompetenzprofil«, S. 98)
Rationalisierung	Rationalisierung	Methodeneinsatz, Reduktion von Nacharbeiten, Software-Familien
Automatisierung	Automatisierung	Entwicklungsumgebungen, Generatoren
Kontinuierliche Verbesserung, TQM	Kontinuierliche Verbesserung	TQM, ISO9000 ff., CMMI, Spice (siehe »Monumentale Modelle«, S. 619), Steuerung über Kenngrößen
Standardisierung	Standardisierung	UML, Plattformen (Windows, Linux), GUIs
Plattformstrategie (z. B. Autoindustrie)	Plattformstrategie	Windows-Plattform, J2EE-Plattform
Modulkomponenten (z. B. Autoindustrie)	Komponentenbasierte Architektur	
Verringerung der Fertigungstiefe	Outsourcing, Einkauf von Komponenten	Prozessmodelle mit Lieferantenbeziehungen, Konzentration auf Kernkompetenzen
Nutzung globaler Märkte	Nutzung globaler Märkte	
Globale Nutzung von Lohngefälle	Offshoring, Nearshoring	Siehe »Globale Softwareentwicklung«, S. 281

Tab. 5.2-1: Industrielle Mittel.

Die **Rationalisierung** führt in der Softwarebranche im Gegensatz zur klassischen Industrie aber *nicht* zur Reduktion der Arbeitsplätze, sondern zur Erstellung immer komplexerer Software.

Beispiele

Ein typisches Handy enthält 2 Millionen Zeilen Programmcode. 2010 wird es 10-mal mehr Software besitzen. General Motors schätzt, dass 2010 jedes ihres Autos 100 Millionen *Lines of Code* enthält. Hinzu kommt, dass Systeme immer mehr in Systeme integriert werden. Ein Luftverkehrsüberwachungssystem hängt von Dutzenden von anderen Netzwerken ab, die Kommunikation-, Wetter-, Navigations- und andere Daten bereitstellen [Charette 05, S. 37 ff.].

Eine zunehmende **Standardisierung** ermöglicht einen Massenmarkt, z. B. die einheitliche Anordnung von Lenkrad, Gas- und Bremspedal bei Autos. Auch die proprietären Standards, z. B. von Microsoft bezogen auf das Betriebssystem, die Office-Programme und die grafische Bedienungsoberfläche, haben sicher zur Verbreitung der PCs und ihrer Software beigetragen.

Ziel einer **Plattformstrategie** ist es, weit gefächerte individuelle Bedürfnisse mit wenigen Basisvarianten zu erzielen [Taubner 05, S. 293]. Beispielsweise basieren die Modelle VW Golf, Audi A3, Skoda

Octavia, Seat Cordoba, Seat Toledo, VW Vento und VW Beetle auf derselben technischen Plattform. Dennoch werden pro Jahr nur zwei vollständig identische VW-Golf-Modelle ausgeliefert. Bei der Softwareerstellung werden Prozessmodelle entwickelt, die es erlauben, Software-Familien zu entwickeln *(software product line engineering)* [Pohl, Böckle, van der Linden 05] (siehe »Das Modell für Produktfamilien/Produktlinien«, S. 547).

Die Verringerung der **Fertigungstiefe** hat in klassischen Industrien zu wesentlichen Veränderungen geführt. Statt Teile selbst zu fertigen, werden Teile bei spezialisierten Lieferanten eingekauft. Das senkt die Preise, verbessert die Qualität und ermöglicht teilweise den Einkauf innovativerer Teilprodukte.

Zitat

»Ein Automobilhersteller zeichnet sich heute nicht mehr primär durch Kompetenz in der Fertigung aus. Selbst Ingenieurskönnen tritt teilweise in den Hintergrund. Erfolgsentscheidend werden die Beherrschung des Gesamtsystems, die Prozessführerschaft und die (emotionale) Markenerscheinung. Auch der Einkauf gewinnt damit an Bedeutung« [Taubner 05, S. 293].

Konsequenzen der Software-Industrialisierung

Die zunehmende Industrialisierung der Softwareerstellung hat folgende Auswirkungen:

- **Kommoditisierung** von Software: Software wird zu einer Konsumware mit standardisierten Produktmerkmalen. Die Bezeichnung geht auf das englische Wort *commodities* zurück, das eine Konsumware bzw. einen Bedarfsartikel beschreibt. Der Weg von Software hin zu einer Konsumware bedeutet, dass die Ware, hier eine Software, von vielen produziert werden kann und dadurch auch die Preise sinken. Kommoditisierung wird durch Standardisierung gefördert. Beispielsweise liegt die Grundfunktionalität von E-Mail-Systemen – insbesondere durch die verfügbaren Protokolle – fest, so dass eine Vielzahl von Anbietern die Grundfunktionalität sogar kostenlos anbieten. Ein anderes Beispiel sind Finanzdienstleistungen. Früher hatte jedes Finanzinstitut seine eigene Software, um Überweisungen abzuwickeln. Es war undenkbar, diese Aufgabe einem konkurrierenden Institut zu übergeben. Heute hat sich die Auffassung durchgesetzt, dass diese Aufgabe kein Alleinstellungsmerkmal eines Finanzinstituts ist. Die Postbank wickelt heute für viele konkurrierende Finanzinstitute die Transaktionen ab.
  Die Abb. 5.2-1 zeigt den Zusammenhang zwischen Kommoditisierung und Standardisierung. Die Softwaretypen wandern von links unten nach rechts oben. Interessant ist, dass in Deutschland 93 Prozent der Beschäftigten in der Softwarebranche im linken Be-

reich arbeiten und nur sieben Prozent im Produktgeschäft (Stand 2007). In anderen Ländern sieht das anders aus. Deutschland ist führend bei individuellen Lösungen, daher der starke Personaleinsatz in diesem Bereich.
Viel diskutiert wird in diesem Zusammenhang die These von Nicholas Carr, der behauptet, dass IT und Software zunehmend als Wettbewerbsfaktoren an Bedeutung verlieren, da sie selbstverständlich, einfach und überall verfügbar werden wie die Eisenbahn und die Elektrizität [Carr 04].

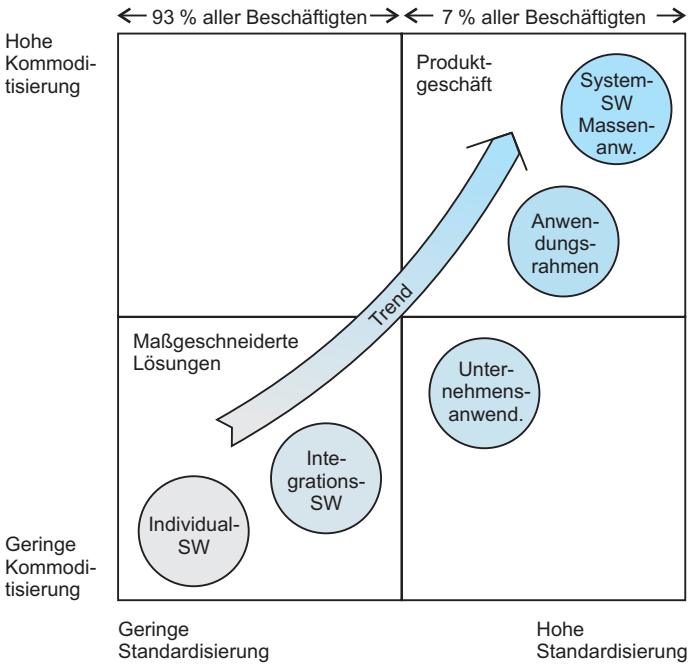


Abb. 5.2-1: Trend zur Standardisierung und Kommoditisierung.

- **Innovation** oder **Rationalisierung**: Als Folge der Kommoditisierung können Unternehmen zwei Wege einschlagen. Durch ständige Innovation können Unternehmen immer komplexere Software erstellen und/oder neue Anwendungsgebiete erschließen. Alternativ kann sich ein Unternehmen darauf konzentrieren, besonders gut in Automatisierung und Rationalisierung zu sein, um damit vorhandene Produkte immer kostengünstiger zu produzieren und anzubieten. Ein Beispiel dafür ist die Pharmaindustrie. Die klassischen Pharmaunternehmen versuchen durch aufwendige Forschung neue Medikamente zu entwickeln. Die Generikahersteller dagegen optimieren ständig ihre Produktionsverfahren, um bekannte Medikamente rationeller herzustellen.

- Produkte erweitert um **Dienstleistungen**: Der Kunde will heute *nicht* ein Produkt kaufen, sondern eine Lösung für sein Problem. Dazu gehört, dass zusätzlich zum Produkt Dienstleistungen angeboten werden, die aus dem Produkt eine Problemlösung machen (*Software as a Service*, SaaS). Die Dienstleistungen gehen über die Anpassung des Produkts an spezifische Gegebenheiten, über Wartung und Pflege des Produkts bis hin zum Betrieb des Produkts beim Anbieter (**ASP**). Noch weitergehender sind Konzepte, die die Bezahlung eines Softwareprodukts nur in Abhängigkeit von der tatsächlichen Inanspruchnahme vorsehen, in Analogie zum Wasser- und Stromverbrauch. Bezahlt werden muss nur die tatsächliche Inanspruchnahme. Konzepte, bei denen Computer- und Softwareleistungen nach Bedarf bezahlt werden, werden als *Utility Computing* oder *Computer on Demand* bzw. *Software on Demand* bezeichnet. In letzter Zeit kommt ein neues Geschäftsfeld dazu: BSP *(Business Service Providing)*. Dabei werden Teilaufgaben des Kunden auf dessen System oder in ein Rechenzentrum übernommen. Beispiel hierfür ist die Übernahme des Energiedatenmanagements für ein Stadtwerk, das aber weiterhin die kaufmännischen Prozesse selbst durchführt.
- **Verändertes Management**: Die Steuerung des Produktionsprozesses von Software erfolgt zunehmend über Kennzahlen (siehe »Maße definieren, einführen und anwenden«, S. 377).
- **Veränderte Infrastruktur**: Datenautobahnen und Server-Farmen werden für die Infrastruktur bestimmend. Serviceorientierte Architekturen (SOA) erlauben eine einfache Einbindung und Komposition standardisierter Funktionsbausteine, die sich dann zu komplexen Services zusammenfügen lassen.
- **Veränderte Unternehmensstrukturen**: Der Konzentrationsprozess durch Fusionen und Übernahmen hält an, siehe z. B. Oracle.

5.3 Das Umfeld eines Softwaremanagers

Aufgabenfeld

Ein Softwaremanager kann heute in einem sehr unterschiedlichen Aufgabenfeld tätig sein:

- Tätigkeit in einem Software-Haus – also auf der **Auftragnehmer**- bzw. Lieferantenseite.
- Erstellung von Standardsoftware für den anonymen Markt.
- Erstellung von Komponenten: Software-Haus ist spezialisiert auf Software-Komponenten.
- Erstellung von Individualsoftware.

- Tätigkeit in einer IT- oder Software-Abteilung in einem Unternehmen – ebenfalls auf der (internen) Auftragnehmer- bzw. Lieferantenseite.
- Unterstützung der Unternehmensziele, in Konkurrenz zu externen Anbietern.
- Tätigkeit in einer Institution, die Softwareaufträge vergibt (intern oder extern) – also auf der **Auftraggeberseite**.
- Auswahl der Lieferanten, Festlegung der Anforderungen, Abnahme und Einführung der Software.

Art der Software

Neben dem Aufgabenfeld spielt die Art der zu entwickelnden Software eine Rolle:

- kaufmännisch-administrativ
- technisch
- eingebettet

Frage

Warum spielt die Art der zu entwickelnden Software eine Rolle für das Management?

Antwort

Die einzelnen Softwarearten haben Auswirkungen auf das Entwicklungs- und Qualitätsmodell sowie auf die Fähigkeiten der Mitarbeiter und die einzusetzenden Werkzeuge. Bei eingebetteter Software oder softwareintensiven Systemen steht die Software nicht allein im Zentrum der Überlegungen, sondern viele Randbedingungen müssen berücksichtigt werden.

Die zunehmende Industrialisierung der Softwareerstellung führt zu einer zunehmenden Bedeutung von Management, Koordination und Konzeption (siehe »Vom Kunsthandwerk zur Industrialisierung«, S. 152).

Software-Auftragnehmer

Lieferanten für Individualsoftware sind durch die Kommoditisierung einem zunehmenden Preisdruck ausgesetzt. Dies zwingt den Lieferanten dazu zu wachsen oder sich auf eine Nische zu konzentrieren. Der Software-Erstellungsprozess muss zuverlässig und kalkulierbar gesteuert werden können. Für einzelne Teile oder Schritte des Erstellungsprozesses müssen die Vorteile der Industrialisierung genutzt werden:

- Einsatz von Werkzeugen
- Nutzung von Spezialisten
- Vergabe von Aufgaben an Unterlieferanten
- Kauf fertiger Komponenten
- Nutzung der Kostenvorteile durch Offshoring

Über folgende Fähigkeiten muss das Softwaremanagement verfügen:

- Projektmanagement und Koordination einschl. räumlich verteilter Projektdurchführung.

- Schnittstellenmanagement
- Abnahme von Zulieferungen
- Qualitätssicherung

Software-Auftraggeber

Auftraggeber für Individualsoftware müssen zwischen folgenden Zielen abwägen:

- preiswert vs. flexibel/schnell
- standardisiert vs. innovativ

Das Softwaremanagement muss über folgende Fähigkeiten verfügen:

- Fähigkeit zum Einkauf und zur Steuerung von Zulieferungen (auch über mehrere Stufen und räumlich verteilt).
- Einsatz von Englisch als durchgehende Projektsprache.

5.4 Zusammenfassung

das Besondere

Softwareentwicklungen benötigen, z. B. verglichen mit Produktionsprozessen, mehr Management, da Software unsichtbar ist, der Entwicklungsfortschritt schwer zu ermitteln ist und es noch unklare Vorstellungen vom richtigen Entwicklungsprozess gibt.

Handwerk vs. Industrie

Software wird unterschiedlich erstellt. Es sind sowohl handwerkliche Verfahren als auch Ansätze einer industriellen Erstellungsweise zu finden. Kennzeichen der Industrialisierung sind Rationalisierung, Standardisierung, Plattformstrategien und verringerte Fertigungstiefen.

Umfeld

Softwaremanager können auf der Auftraggeber- oder der Auftragnehmerseite in Softwarehäusern tätig sein oder in firmeninternen IT-Abteilungen oder Organisationen auf der Auftragnehmerseite arbeiten. Sie können kaufmännisch-administrative, technische oder eingebettete Software als Individual- oder Standardsoftware entwickeln.

http://www.springer.com/978-3-8274-1161-7

Lehrbuch der Softwaretechnik: Softwaremanagement
Balzert, H.
2008, XIX, 721 S., Hardcover
ISBN: 978-3-8274-1161-7