Industrielle Softwareentwicklung

Leitfaden und Orientierungshilfe

BITKOM

Impressum

Herausgeber: BITKOM
Bundesverband Informationswirtschaft,
Telekommunikation und neue Medien e. V.
Albrechtstraße 10 A
10117 Berlin-Mitte
Tel.: 030.27576-0
Fax: 030.27576-400
bitkom@bitkom.org
www.bitkom.org

Ansprechpartner: Stephan Ziegler
Tel.: 030.27576-243
s.ziegler@bitkom.org

Autoren: Werner Achtert, TÜV Informationstechnik GmbH, Torsten Becker, BESTgroup Consulting & Software GmbH, Hubert Biskup, IBM Deutschland GmbH, Dirk Hellebrand, FOURTH PROJECT GmbH, Jürgen Herczeg, T-Systems International GmbH, Stephan Krause, CSC Deutschland Solutions GmbH, Marco Kuhrmann, Technische Universität München, Frank Maar, Microsoft Deutschland GmbH, Frank Marschall, T-Systems International GmbH, Matthias Minich, T-Systems International GmbH, Frank Simon, SQS Software Quality Systems AG, Stephan Ziegler, BITKOM e.V.

Redaktion: Anne Müller (BITKOM)

Gestaltung / Layout: Design Bureau kokliko / Anna Müller-Rosenberger (BITKOM)

Industrielle Softwareentwicklung

Leitfaden und Orientierungshilfe

BITKOM

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Executive Summary

Die Bedeutung von Software als Basistechnologie für Innovation und Wertschöpfung im 21. Jahrhundert nimmt stetig zu. Innovative Produkte und Dienstleistungen mit Wertschöpfung in Hochlohnländern sind ohne Software kaum mehr möglich.

Während zu Beginn des IT-Zeitalters die Entwicklung von IT-Systemen noch als ausschließlich kreativer Akt den Fokus auf der grundsätzlichen Machbarkeit hatte, gewinnt aktuell ein industrialisiertes Vorgehen (Software Engineering) immer mehr an Bedeutung, um die IT in das Spannungsfeld von Effektivität und Effizienz zu integrieren. Die damit erwartete Garantie von Qualität und Produktivität ist die Voraussetzung für Wertschöpfung und für Arbeitsplätze in der IT-Industrie in Hochlohnländern wie Deutschland.

In diesem Leitfaden wird dargelegt, wie eine derartige Industrialisierung der IT konkret aussehen kann, welche unterschiedlichen Aspekte bereits erfolgreich umgesetzt werden und welche kurzfristigen Ergebnisse noch zu erwarten sind. Hierfür werden umfangreiche Erfahrungen aus anderen Industrialisierungsbereichen (z. B. Maschinenbau) analysiert, um daraus zusammen mit den bereits erfolgreichen Bereichen fünf Industrialisierungsdimensionen zu extrahieren, anhand derer sich IT-Industrialisierung festmachen lässt. Diese fünf Industrialisierungsdimensionen sind:

- Standardisierung
- Automatisierung
- Wiederverwendung
- Spezialisierung
- kontinuierliche Verbesserung

Die IT kann bereits heute in allen fünf Dimensionen praxistaugliche Ergebnisse vorweisen: Diese Industrialisierungsansätze und -methoden umfassen dabei unterschiedliche Typen von Frameworks, Prozessstandards, hoch automatisierte Entwicklungsumgebungen, Testautomatisierung, domänenpezifische Sprachen (DSLs), komponentenorientiertes Vorgehen, Produktlinien und Feedback-Loops.

Diese bereits existierenden Industrialisierungsmethoden werden jeweils in den oben aufgeführten fünf Dimensionen verortet und anschließend detailliert in der dominierenden Dimension bzgl. der Industrialisierungsauswirkungen beschrieben.

Die Analyse anderer Domänen zeigt allerdings auch, dass Industrialisierung mehr ist als das Zusammenstecken von Industrialisierungsfragmenten. Vielmehr zeigt dieser Leitfaden immer wieder notwendige Vor- und Nachbedingungen (Constraints) auf, deren Nichtbeachtung durchaus zu negativen Effekten bei der Umsetzung einer IT-Industrialisierung führen kann. Eine erfolgreiche Industrialisierung besteht demnach nicht in der Optimierung einzelner Industrialisierungsdimensionen, sondern in der ganzheitlichen, gleichwertigen Orchestrierung der unterschiedlichen Maßnahmen und Methoden aus den einzelnen Dimensionen hin zu einer umfassenden Industrialisierung in der Softwareentwicklung.

Der Leitfaden schließt mit einer kurzen Roadmap, in dem die Orchestrierung als Orientierung beschrieben wird, die es jedem Unternehmen erlaubt, sich auf die erfolgversprechende Reise der IT-Industrialisierung zu begeben. Nach der Erfüllung spezifischer Mindestanforderungen hilft dieser Leitfaden, die bereits heute existierenden Industrialisierungsmethoden harmonisch in einen effektiven und effizienten Softwareentwicklungsprozess zu überführen. Gescheiterte Industrialisierungskonzepte, über die immer wieder berichtet wird, müssen demnach nicht zwangsläufig dem Konzept selbst geschuldet sein, sondern können nur auf die Nichterfüllung der Vorbedingungen hinweisen. Daher beschreibt dieser Leitfaden spezifische Qualitätsprüfpunkte (sogenannte Quality Gates), die im Idealfall vor der Umsetzung von Industrialisierungskonzepten erfüllt sein müssen. Wird die Aussage unterstellt, dass neue Liefermodelle in der

Softwareindustrie – wie beispielsweise Cloud Computing – ausgereift sind und die erfolgreichste Form der IT-Industrialisierung darstellen, so liefert dieser Leitfaden hochaktuelle Informationen, die helfen sollen, entsprechende Angebote zu entwickeln und am Markt zu platzieren. Damit unterstützt die Publikation, dass dem Standort Europa auch zukünftig eine gewichtige Rolle im globalen Wettbewerb zugeordnet wird.

1 Einleitung und Motivation

Die IT-Branche ist ein vergleichsweise junger und sehr schnelllebiger Wirtschaftssektor. Ungeachtet dessen gelten auch für Informationstechnologien zunehmend die aus anderen Industrien bekannten Marktmechanismen wie beispielsweise Verdrängung von Wettbewerbern über den Preiskampf und eine möglichst schnelle Time-to-Market. Ziel vieler Softwareanbieter ist es daher, ein möglichst hochwertiges Produkt mit möglichst geringen Produktions- bzw. Entwicklungskosten zu erstellen, d. h., die Effizienz rückt zunehmend in den Vordergrund. Die Reproduktion eines immateriellen Gutes wie Software spielt aus Kostensicht dabei eine untergeordnete Rolle. Die Produktentwicklung selbst – also die Softwareerstellung – hingegen ist bis heute kostenintensiv und nur selten exakt zu kalkulieren und erfüllt damit nicht die Forderung nach gesteigerter Effizienz.

Die Entwicklung eines ingenieurmäßigen Vorgehens auf der einen Seite und die Verlagerung von Softwareentwicklung in Länder mit deutlich geringeren Lohnkosten auf der anderen Seite sind die Folge dieser Effizienzfokussierung. Sie sind gleichzeitig Indizien für eine einsetzende Industrialisierung der Softwareindustrie.

Eine Vielzahl der traditionellen Branchen ist hinsichtlich der Industrialisierung – auch mit Unterstützung der Informationstechnologie – jedoch erheblich weiter. Diese Aussage gilt für Sektoren der Investitionsgüterindustrie, beispielsweise für den Automobil- oder Maschinenbau, aber auch für die forschungsintensive Pharmaindustrie. Der Trend geht hier zu Massenprodukten, welche durch den Aufbau aus standardisierten Komponenten in einer Vielzahl von Varianten ein breites Spektrum an Anforderungen abdecken.

Warum lassen sich die Erfahrungen und Konzepte der anderen bereits erfolgreich industrialisierten Branchen nicht 1:1 auf die Informationstechnologie übertragen? Ist die schnell wachsende IT-Branche, die weltweit aus größtenteils akademisch ausgebildeten Spezialisten und deren kreativen Leistungen besteht, nicht bereit, von anderen zu lernen? Nein, an der Bereitschaft mangelt es sicher nicht. So liefert die IT-Industrie eine ständig wachsende und teilweise unüberschaubare Palette an Methoden und Vorgehensmodellen, technologischen Plattformen und Werkzeugen. Darüber hinaus existiert eine Vielzahl von teilweise konkurrierenden Standards und sogenannten Quasi-Standards, welche von jeweils dominierenden IT-Anbietern in spezifischen Sektoren fokussiert werden.

Der Grad der Anwendung dieser Industrialisierungsbausteine variiert allerdings enorm: Die Produkte dieser Branche sind sehr unterschiedlich. Angefangen bei eingebetteter Software (Embedded Software) über komplexe Betriebssysteme und Anwendungssoftware bis hin zu ganzen Netzwerken von Softwarepaketen reicht das Spektrum. Darüber hinaus existieren historisch bedingt Unmengen Individualsoftware, d. h. echte Einzelstücke, die sich nur schwer mit Industrialisierungskonzepten ersetzen lassen.

Diese Vielfalt spiegelt sich auch in der Vielfalt der Einsatzgebiete von IT-Systemen wider: in wenigen Jahren vom Rechenknecht zur weltumspannenden Kommunikationsmaschinerie bis hin zu dem zentralen Informationsmedium auf der Erde – nicht zu vergessen die unzähligen betriebswirtschaftlichen Systeme, ohne welche eine globalisierte Weltwirtschaft sicher nicht möglich wäre.

Die Evolution dieser unterschiedlichen IT-Systeme war und ist primär immer noch geprägt von häufigen technologischen Umbrüchen, die allerdings im Kern alle als unterschiedliche Beispiele in den aufgeführten Industrialisierungsdimensionen kategorisiert werden können: neue Programmiersprachen und IT-Plattformen, verbesserte Kommunikationsprotokolle, explosionsartige Leistungssteigerung der Hardware, ein weltumspannendes Netzwerk, um nur einige zu nennen. Im Automotive-Sektor steht mit der Einführung von alternativen Antrieben erstmals ein vergleichbarer Umbruch gerade bevor, ohne allerdings deren hohen Industrialisierungsgrad infrage zu stellen.

Allen Unwägbarkeiten zum Trotz zwingen die Gesetze des Marktes die Softwarehersteller weltweit, ihre Entwicklungsmethoden zu verbessern; d. h. grundsätzlich über weitere Industrialisierungen nachzudenken. Entlang der folgenden Industrialisierungsmerkmale lassen sich auch für die Softwareentwicklung vielversprechende Ansätze ableiten:

- Qualitätssteigerung
- Hebung von Synergien
- Arbeitsteilung
- Kostenreduktion
- verkürzte Release-Zyklen
- erhöhte Innovationsgeschwindigkeit (Time-to-Market)

Einige erprobte Ideen aus anderen Branchen lassen sich übertragen. Aufgrund der skizzierten Unterschiede zwischen IT-Branche und anderen bereits weiter industrialisierten Industriesektoren weisen die etablierten Ansätze aber beispielsweise erheblich abweichende Kosten/Nutzen-Verhältnisse bei der Anwendung auf Softwareentwicklung auf. Hier müssen neue Wege gefunden werden. Industrialisierung soll nicht zum Selbstzweck stattfinden!

Der vorliegende Leitfaden gibt einen Überblick über die aktuellen Industrialisierungsansätze für die Softwareentwicklung und unterstützt bei der Identifikation von Verbesserungspotenzialen. Außerdem wird herausgearbeitet, dass der Einsatz von Industrialisierungsansätzen bestimmte Vorbedingungen hat, liegen diese nicht vor, wird Industrialisierung nicht die gewünschten Ziele bringen. Auch hier kann die IT aus anderen Bereichen lernen.

Der Fokus dieses Leitfadens liegt daher auf der möglichst effizienten Erstellung von Software bis zu deren Bereitstellung bzw. dem Beginn der produktiven Nutzung unter Verwendung von Industrialisierungskonzepten. Zeitlich nachgelagerte Themen im Kontext des sogenannten Software Lifecycle wie Support, Wartung und der operative IT-Betrieb sind nicht Bestandteil des Leitfadens.

1.1 Begriffsklärung SW-Industrialisierung und SW-Engineering

1.1.1 Industrialisierung

Allgemein:
Die Industrialisierung bezeichnet einen volkswirtschaftlichen Prozess, der gekennzeichnet ist durch eine signifikante Zunahme der gewerblichen Gütererzeugung (sekundärer Sektor) auf Kosten des Agrarbereichs (primärer Sektor). Diese Erzeugung von gewerblichen Massengütern erfolgt mit wachsendem Maschineneinsatz in großgewerblicher, arbeitsteiliger Produktionsorganisation. [Gabl10]

1.1.2 IT-Industrialisierung

IT-Industrialisierung bezeichnet die Automatisierung und Standardisierung des IT-Leistungserbringungsprozesses durch Übertragung bewährter Methoden und Prozesse aus dem Bereich der industriellen Fertigung. Etwas weiter gefasst geht es um die Übertragung typischer, aus Industrieunternehmen stammender Konzepte auf verschiedene Bereiche der IT-Branche, z. B. auf die Hardware- und Softwareentwicklung, aber zunehmend auch auf das Informationsmanagement.

Ziel der Industrialisierung des Informationsmanagements ist die Steigerung von Effizienz und Effektivität der betrieblichen IT-Bereiche sowie etwaig existierender externer IT-Dienstleister. Als typische Beispiele für übertragbare Industrialisierungskonzepte lassen sich folgende anführen:

- Automatisierung
- Standardisierung von Prozessen und Produkten
- Komponentenorientierung und Modulbauweise
- Plattformstrategien und Wiederverwendung
- kontinuierliche Verbesserung verbunden mit einem Streben nach Mess- und Steuerbarkeit
- Arbeitsteilung
- Reduktion der Fertigungstiefe sowie eine globale Beschaffung

[FrSto7]

1.1.3 Charakter der IT-Industrialisierung

Die IT-Industrialisierung lässt sich bzgl. der folgenden vier Grundcharakteristika beschreiben:

- Standardisierung und Automatisierung: Die Herstellkosten lassen sich enorm verringern, indem sowohl die Produkte als auch die Prozesse standardisiert werden. Die Standardisierung der IT lässt sich z. B. anhand der weitverbreiteten Prozessstandards wie CMMI, ITIL, CobiT erkennen. Automatisierungsansätze in verschiedenen Teildisziplinen der Softwareentwicklung ermöglichen zunehmend eine schnellere Produktentwicklung sowie eine verbesserte und effizientere Qualitätskontrolle. Beispiele dafür sind die modellgetriebene Entwicklung mit anschließender Codegenerierung und die Testautomatisierung.
- Modularisierung: Eine modul- und komponentenorientierte Herstellung erlaubt es, Produkte trotz standardisierter und damit kostengünstiger Prozessabläufe zu individualisieren. So basieren im Automobile-Bereich beispielsweise die Modelle VW Golf, Audi A3, Skoda Octavia, Seat Cordoba, Seat Toledo, VW Vento und VW Beetle alle auf derselben Plattform. Mehr noch: Derzeit rollen pro Jahr nur zwei komplett identische VW-Golf-Modelle vom Montageband. Die Modularisierung der IT lässt sich z. B. anhand folgender Virtualisierungstechniken belegen: Grid-, Blade- oder Utility-Computing, SOA, MashUps, Application Server.
- Kontinuierliche Verbesserung: Mithilfe von Qualitätskonzepten – angefangen beim Deming-Cycle über Kaizen und Total Quality Management (TQM) bis zu Six Sigma –versuchen Unternehmen, ihre Produktionsabläufe ständig zu verbessern. Gleichzeitig streben sie danach, die Prozessverbesserung mess- und steuerbar zu machen. Die kontinuierliche Verbesserung der IT lässt sich z. B. anhand teilweise vertragsrelevanter Bestandteile belegen, Beispiele sind Service-Level-Agreements (SLAs), Operative-Level-Agreements (OLAs), Six-Sigma-Konzepte oder Maturity Models (z. B. CMMI).
- Konzentration auf Kernkompetenzen: Um wettbewerbsfähig zu bleiben, haben Unternehmen im vergangenen Jahrhundert ihre Wertschöpfungstiefe branchenübergreifend verringert. Während dies anfänglich noch lokal erfolgte (durch die Gründung von lokalen Tochterunternehmen oder die Einbindung von On-Site-Centres), ist dieses Vorgehen heute am deutlichsten im IT-Outsourcing-Geschäft und im Bereich der Off-Shore-Entwicklung zu beobachten.

1.1.4 Vergleich IT-Industrialisierung und Fertigungsindustrialisierung

Zusammenfassend kann damit das folgende Bild erstellt werden: (vgl. [Breno7])

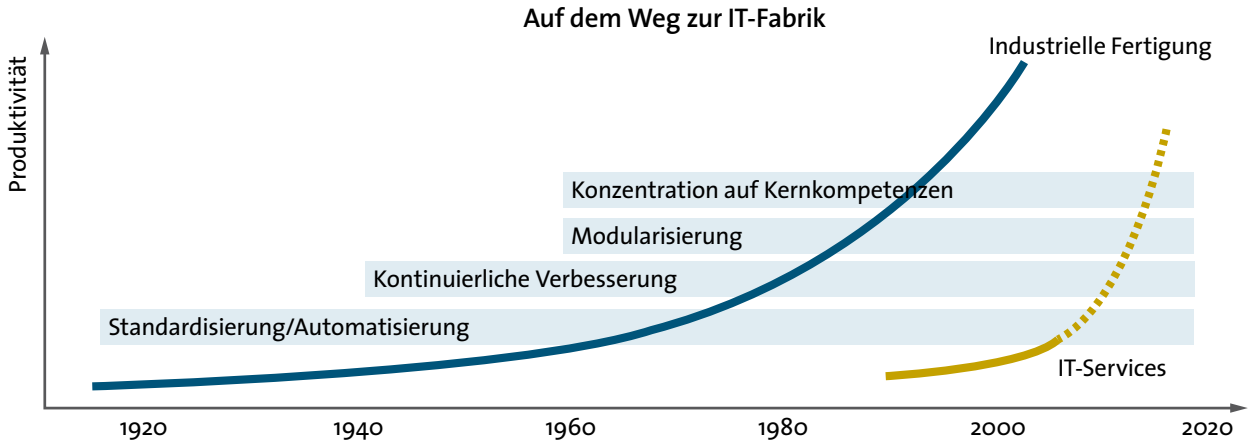


Abbildung 1: Auf dem Weg zur IT-Fabrik; Quelle: Universität St. Gallen

Demnach ist die Entwicklung der Industrialisierung durchaus mit der in der IT und deren Services vergleichbar (vgl. horizontale graue Balken). Allerdings wird daran auch deutlich, dass die Industrialisierung im IT-Bereich noch nicht entsprechend vorangekommen ist und heute der industriellen Fertigung rund 20 Jahre nachhängt. Dies hat für die IT allerdings auch Vorteile, da sie auf bereits etablierte Best Practices im industriellen Fertigungsbereich schauen kann und „nur" noch deren Migration auf die IT-Services andenken und durchführen muss.

1.1.5 Weitere Übertragungen typischer Industrialisierungsansätze auf IT

Ein weiterer interessanter Aspekt der Industrialisierung wird für die nicht IT-spezifischen Bereiche als Deindustrialisierung bezeichnet:

Deindustrialisierung bezeichnet eine strukturelle Veränderung der Wirtschaftsgefüge, die in entwickelten Industrienationen verortet ist. Sie findet ihren Ausdruck in der (relativen) Abnahme der Bedeutung des produzierenden Gewerbes und zeichnet sich durch absolutes Absinken der Beschäftigung, der Produktion, der Rentabilität, des Kapitalstocks, absoluter Rückgang der Exporte und langlebige Handelsdefizite aus. Insbesondere die Globalisierung gilt als Treiber dieser Entwicklung.

Vielmehr gibt es zwischen Industrie- und Dienstleistungssektor eine (Wechsel-)Beziehung. Sie findet u. a. ihren Ausdruck in der Drei-Sektoren-Hypothese nach Jean Fourastié. In dieser unterteilt er die Produktionsstrukturen einer Volkswirtschaft in drei Sektoren: den primären Sektor (Landwirtschaft), den sekundären Sektor (Industrie und Handwerk) sowie den tertiären Sektor (Dienstleistungen).

Fourastié zufolge würden Wirtschaft und Gesellschaft sich entlang dieser Sektorenaufteilung entwickeln; vom primären Sektor zum sekundären und von dort zum tertiären Sektor, „[...] oder mit vereinfachenden Worten: Agrargesellschaften verwandeln sich zunächst in Industriegesellschaften und Industriegesellschaften schließlich in Dienstleistungsgesellschaften. [...] Die Schwerpunktverlagerung hin zum tertiären Sektor ist mit wichtigen Veränderungen in der Sozialstruktur, im Schichtgefüge und in den Lebens- und Arbeitsbedingungen verknüpft." [Grab98]

1.1.6 Software Engineering: Einbindung in IT-Industrialisierung

"Software engineering is the application of a systematic, disciplined, quantifiable approach to the development, operation, maintenance, and retirement of software, that is, the application of engineering to software." [IEEE729]

Die Schlüsseltugenden eines ingenieurmäßigen Vorgehens, insbesondere für den Bereich der Softwareerstellung, Wartung und Ersetzung, sind folglich:

- Das Vorgehen ist systematisch und wiederholbar.
- Das Vorgehen ist diszipliniert.
- Das Vorgehen muss quantifizierbar sein.

„Die Wettbewerbsfähigkeit der deutsche Wirtschaft [...] [hängt] entscheidend von der Fähigkeit [ab], Softwareintensive Produkte und Dienstleistungen mit höchster Qualität zu erstellen. Software Engineering über dem Weltniveau ist die Voraussetzung dafür, dass Deutschland seine führende Stellung im Ingenieurbereich, etwa im Maschinenbau, halten und ausbauen und entsprechende Positionen in neuen Sparten, etwa im modernen Gesundheitswesen (e-Health) aufbauen kann." [Broy+06]

Viele Prinzipien des ingenieurmäßigen Vorgehens lassen sich als Hilfsmittel zur Erreichung einer IT-Industrialisierung darstellen. Das bedeutet: Industrialisierung mittels ingenieurmäßigem Vorgehen!

Die konsequente Anwendung und Nutzung der Industrialisierungsprinzipien hat im vergangenen Jahrhundert die meisten Branchen grundlegend verändert. Die relativ junge Disziplin der IT hat diesen Wandel gerade erst begonnen und steht in den nächsten Jahren vor wesentlichen Veränderungen. Wer nicht als Verlierer daraus her-

vorgehen will, muss die Bedeutung der Industrialisierung in der IT gründlich durchdenken.

„Deutschland ist heute Weltmeister in Branchen wie Elektrotechnik und Automobilbau. Das liegt an der einzigartigen Innovationsleistung von Ingenieurinnen und Ingenieuren." Bundesforschungsministerin Bulmahn beim Startschuss für das Jahr der Technik 2004

1.2 Industrialisierung von Softwareentwicklung

Offenkundig kann die IT für eine bessere Effizienz und Effektivität Ideen der Industrialisierung in anderen Bereich positiv aufnehmen. Auch wenn es überraschend erscheinen mag, dass die IT diesbezüglich gegenüber anderen Branchen Nachholbedarf hat, so ergibt sich daraus doch heue eine große Chance: Aus den Erfahrungen (negative wie positive) zu lernen, um die IT auf direktem Weg ein Stück nach vorne zu bringen. Die Übertragung der industriellen Ansätze sollte hierbei den folgenden drei Punkten genügen:

- Die Industrialisierung sollte im Bereich der IT ingenieurmäßig erfolgen. Dafür bedarf es einer Systematik, einer entsprechenden Disziplin und einer Quantifizierung. Industrialisierung ist damit kein Selbstzweck, sondern muss sich – auf Basis der ermittelten Kennzahlen – bezüglich der Erreichung gesteckter Ziele fortwährend rechtfertigen. Auch der häufig wahrgenommene Trend, jedem neuen technischen Hype aufzuspringen, sollte durch ein ingenieurmäßiges Vorgehen geprüft werden. Die zugrunde liegende Systematik ist dabei vermutlich unternehmensweit vorzugeben: Hierzu gehört eine systematische Portfolioanalyse und das bewusste Setzen von Kernkompetenzen innerhalb des Unternehmens sowie anschließend das Identifizieren von Automatisierungs-, Standardisierungs- und auch Auslagerungsmöglichkeiten.
- Deindustrialisierung: Ein Industrialisierungstrend sollte – und hier kann aus anderen Bereichen wieder effektiv gelernt werden – auch die Möglichkeit der Deindustrialisierung berücksichtigen. Es kann also durchaus richtig sein, für einzelne Bereiche von den Industrialisierungscharakteristika zu abstrahieren und stattdessen entsprechende Dienstleistungen als „Blackbox" zu verwenden. Auch diese Entscheidung sollte allerdings ingenieurmäßig erfolgen, was wiederum zu einer fortwährenden Hinterfragung der Wirtschaftlichkeit führt. Neue Trends wie Cloud Computing oder SaaS sind also durchaus als Industrialisierung alias Deindustrialisierung zu verstehen.
- Auf die wesentlichen Charakteristika der Industrialisierung aus 1.1.2 – Automatisierung und Standardisierung, Modularisierung, kontinuierliche Verbesserung und Reduktion auf die Kernkompetenzen – und deren Übertragung auf die Softwareentwicklung wird in Kapitel 3 ff. weiter detailliert eingegangen.

Will die IT Industrialisierung gewinnbringend einsetzen, so sollte sie diese drei Bereiche synchronisieren, wie in der folgenden Abbildung noch einmal dargestellt:

Erfolgreiche IT-Industrialisierung ist bedingt durch das Ineinandergreifen verschiedener Komponenten.

Abbildung 2: Aspekte erfolgreicher IT-Industrialisierung

2 Einordnung der Industrialisierungsansätze für die Softwareentwicklung

In den vergangenen 20 Jahren sind verschiedene Ansätze zur Industrialisierung der Softwareerstellung entwickelt worden. Alle Ansätze zielen auf eine effektivere und effizientere Produktentwicklung ab, unterscheiden sich aber teilweise erheblich in Bezug auf die primären Stellschrauben und Schwerpunkte der Industrialisierungsbemühungen. Einige Industrialisierungsansätze betreffen beispielsweise primär die organisatorischen Strukturen, wohingegen andere auf das Produkt oder den Entstehungsprozess fokussiert sind. In Abbildung 3 sind einige Industrialisierungsmaßnahmen anhand der drei wesentlichen Anwendungsebenen strukturiert.

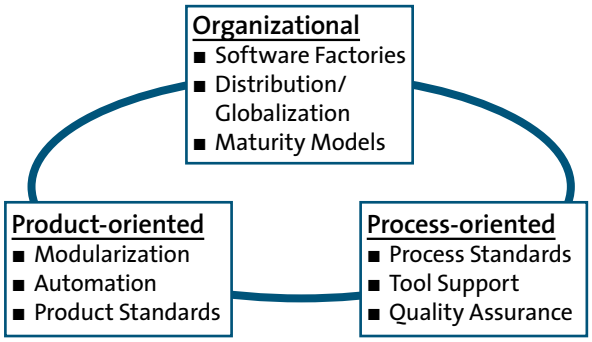


Abbildung 3: Industrialisierungsmaßnahmen auf drei Anwendungsebenen

Bei produktorientierten Ansätzen findet die Industrialisierung am erstellten Softwareprodukt selbst statt. Dazu zählen beispielsweise:

- Einsatz von Softwarebibliotheken und Frameworks
- Einsatz von Standard-Software-Stacks
- komponentenbasierte Softwareentwicklung
- Service-orientierte Softwarearchitekturen (SOA)
- automatische Generierung von Software aus fachlichen und/oder technischen Modellen (modellbasierte Ansätze) – gleichzeitig produkt- und prozessorientierter Ansatz

Bei einer prozessorientierten Industrialisierung wird entlang der Wertschöpfungskette von der Anforderungsanalyse und Konzeption über die Realisierung bis zur Einführung optimiert. Beispiele dafür sind:

- Anwendung standardisierter Prozesse und Vorgehensmodelle
- Einsatz von Richtlinien, Templates, Checklisten
- Entwicklung mithilfe standardisierter Werkzeugketten: Modellierungswerkzeuge, integrierte Entwicklungsumgebungen (IDEs), automatisierte Build-Prozesse
- Einsatz von prozessbegleitenden Qualitätssicherungsmaßnahmen: Reviews, Tests, Projektmeilensteine zur Qualitätssicherung (Quality Gates)
- prozessbegleitende KPIs und Softwaremetriken

Weitere Beispiele für die Industrialisierung mit Fokus auf die Organisation sind:

- Einsatz spezialisierter Entwicklungsteams (fachlich oder technisch) bis zu organisierten Software Factories
- verteilte Softwareentwicklung (inkl. Near-/Offshore-Entwicklung)
- Implementierung von organisatorischen Reifegraden (Maturity Models, z. B. CMMI), wobei die dabei etablierten Methoden auch prozessorientiert wirken (siehe hierzu auch weiter unten: Einordnung der Industrialisierungsansätze)
- Etablierung eines kontinuierlichen Verbesserungsprozesses und der Wiederverwendung von Best Practices im Unternehmen

Die organisations-, produkt- und prozesszentrierte Klassifikation der Methoden und Ansätze zur industriellen Softwareentwicklung fokussiert jeweils auf die primär davon betroffenen Bereiche. Wichtig ist aber, dass alle in den Kapiteln 3 bis 6 im Detail beschriebenen Industrialisierungsansätze in der Regel auch Anpassungen in allen anderen Bereichen mit sich bringen. Beispielsweise ist bei der Einführung verteilter Entwicklung bzw. der Offshore-Entwicklung neben der veränderten Organisationsstruktur der Prozess inklusive der Übergabepunkte für Entwicklungsartefakte häufig neu zu gestalten.

Ein vergleichbares Beziehungsgeflecht ergibt sich, wenn die bekannten Methoden und Ansätze zur industriellen Softwareentwicklung auf die in Kapitel 1 bereits vorgestellten Dimensionen der Industrialisierung abgebildet werden. Tabelle 1 gibt einen Überblick darüber, wie stark die Methoden und Ansätze die Industrialisierungsdimensionen – Standardisierung, Automatisierung, Wiederverwendung, Spezialisierung und kontinuierliche Verbesserung – unterstützen.

Die Matrix zeigt, dass fast alle der unterschiedlichen Methoden jeweils mehrere Industrialisierungsdimensionen betreffen. Komponentenbasierte Softwareentwicklung beispielsweise hat das primäre Ziel, die entwickelten Bausteine möglichst häufig wiederzuverwenden. Damit einher geht aber immer die Standardisierung der Schnittstellen und der entsprechenden Funktionen. Die Nutzung von durchgängigen Werkzeugketten von der Anforderungsanalyse bis zum abschließenden Test des Produktes deckt hingegen verstärkt die Dimension der Automatisierung als auch der Spezialisierung ab.

Wichtig bei allen Industrialisierungsbemühungen für die Softwareentwicklung ist eine ganzheitliche Betrachtung der Möglichkeiten. Die jeweils individuelle Auswahl an Methoden und Ansätzen zielt immer auf mehrere Industrialisierungsdimensionen und betrifft in der Regel sowohl die Organisation, den Entwicklungsprozess sowie das Produkt bzw. dessen Komponenten. Industrialisierung als etwas Triviales, Monodimensionales aufzufassen, ist daher nicht zielführend.

In den folgenden Kapiteln 3 bis 6 werden unterschiedliche Methoden und Ansätze im Einzelnen vorgestellt. Sie sind anhand ihrer primären Industrialisierungsdimensionen – Standardisierung, Automatisierung, Wiederverwendung und Spezialisierung – strukturiert. Anschließend wird im Kapitel 7 „Continuous Improvement“ noch einmal auf das Zusammenspiel zwischen Organisationsstruktur, Entwicklungsprozess und dem Produkt selbst eingegangen.

Dimensionen → Werkzeuge/Methoden ↓	Standardisierung	Automatisierung	Wiederverwendung	Spezialisierung
Standardisierte Vorgehensmodelle und Frameworks	Abschnitt 3.1			
Verwendung von Standards	Abschnitt 3.2			
Automatisierung durch geeignete Werkzeuge		Abschnitt 4.1		
Domänenspezifische Sprachen und modellgetriebene Softwareentwicklung		Abschnitt 4.2		
Testautomatisierung		Abschnitt 4.3		
Komponentenbasierte Softwareentwicklung			Abschnitt 5.1	
Entwicklungslinien und Disziplinen				Abschnitt 6.1
Software-Produktlinien				Abschnitt 6.2
Verteilte Entwicklung und intelligentes Sourcing				Abschnitt 6.3

primäre Industrialisierungsdimension
zusätzliche Unterstützung der Industrialisierungsdimension
kaum Einfluss auf Industrialisierungsdimension

Tabelle 1: Matrix – Einordnung Werkzeuge und Methoden in die SE-Industrialisierungsdimensionen

3 Schwerpunkt Standardisierung

3.1 Standardisierte Vorgehensmodelle und Frameworks

3.1.1 Einführung

Was sind Vorgehensmodelle?

Vorgehensmodelle (oder Prozessmodelle) in der Softwareentwicklung regeln den Prozess der Entwicklung von Softwaresystemen. Sie legen hierfür Ergebnistypen fest, die im Rahmen der Entwicklung erstellt werden sollen und geben Aktivitäten und gegebenenfalls Meilensteine zu deren Erstellung vor.

Vorteile von Vorgehensmodellen

Softwareunternehmen können in vielerlei Hinsicht von standardisierten Prozess- und Vorgehensmodellen profitieren. So stellen einheitliche Vorgehensmodelle die Grundlage für einen gemeinsamen Sprachgebrauch und ein damit einhergehendes gemeinsames Verständnis der jeweiligen Abläufe dar. Unternehmensweite Vorgehensmodelle erlauben die systematische Wiederverwendung bewährter Best Practices, machen Projekte vergleichbar, verkürzen die Einarbeitungszeit von Mitarbeitern in neuen Projekten etc.

Vorgehensmodelle unterscheiden sich oft stark in ihrem Detaillierungsgrad. Im Gegensatz zu Prozessdefinitionen in der Fertigungsindustrie werden jedoch nie starre Abläufe festgelegt, wie z. B. die Montage eines Motorblocks, die sich dann genau so tausend- oder millionenfach wiederholen. Stattdessen müssen Vorgehensmodelle in der Softwareentwicklung dem Umstand Rechnung tragen, dass jedes Entwicklungsprojekt unterschiedlich ist, auch wenn es zwischen einzelnen Projekten durchaus Gemeinsamkeiten gibt. Dementsprechend geben Vorgehensmodelle lediglich einen einheitlichen Rahmen für Entwicklungsvorhaben vor und bieten bewährte Lösungen für wiederkehrende Problemstellungen an.

3.1.2 Spezialisierung vs. Standardisierung und Wiederverwertung

In großen Softwarehäusern mit zahlreichen unterschiedlichen Projekten zeigt sich oft sehr deutlich, dass ein einziges, monolithisches Vorgehensmodell nicht den Anforderungen aller Projekte gerecht werden kann. Eine Vielzahl unterschiedlicher Projekte erfordert jeweils angepasste Vorgehensweisen. So benötigt beispielsweise ein SAP-Projekt andere Entwicklungsmethoden als ein Projekt zur Individualentwicklung einer Java-Anwendung. Zahlreiche weitere Faktoren wie z. B. das Anwendungsumfeld, eingesetzte Technologien, Projektumfang und -dauer, Grad der erforderlichen Agilität etc. haben weiteren Einfluss auf die Anforderungen eines Projekts an einen Entwicklungsprozess, der den Projektbedürfnissen gerecht wird. Dementsprechend werden meist mehrere angepasste Vorgehensmodelle für die wichtigsten Projekttypen einer Organisation benötigt.

Wünschenswert: Hoher Grad an Wiederverwendung

Während in vielen Bereichen auf das jeweilige Projektumfeld spezialisierte Methoden benötigt werden, gibt es andere Methodenbausteine, die sich in sehr vielen Kontexten wiederverwenden lassen. Zum Beispiel sind Methoden und Ergebnistypen des Systementwurfs oft abhängig von der jeweils eingesetzten Realisierungstechnologie. Andererseits sind Techniken zum Review von Ergebnissen (nahezu) universell einsetzbar. Es liegt im Interesse jedes Unternehmens, einen möglichst hohen Grad an Wiederverwendung von Vorgehensbausteinen zu erreichen. Dies erspart zum einen den Aufwand einer redundanten Definition und Pflege solcher Vorgehensbausteine. Zum anderen erhöht deren Wiederverwertung den Grad der Standardisierung im Unternehmen. Innerhalb der T-Systems SI (Systems Integration) gibt es beispielsweise vier Ausprägungen von Reviews, die allen Mitarbeitern bekannt sind, und in allen Arten von Projekten Verwendung finden können.

3.1.3 Von Best Practices zum projektspezifischen Vorgehen

Es gilt also, einerseits von der Wiederverwendung von im Unternehmen entstandenen Standards und Best Practices zu profitieren und andererseits Projekten Vorgehensweisen anzubieten, die jeweils an ihre konkrete Lage und Problemstellung angepasst sind. Hierzu wird innerhalb der T-Systems SI ein mehrstufiges Verfahren etabliert, welches das Vorgehen – vom Aufsammeln von Best Practices in der Organisation bis hin zum projektspezifischen Vorgehensmodell in einzelnen Projekten – abdeckt.

Das Vorgehen ist in Abbildung 4 grafisch skizziert. Die einzelnen Schritte sind im Anschluss erläutert.

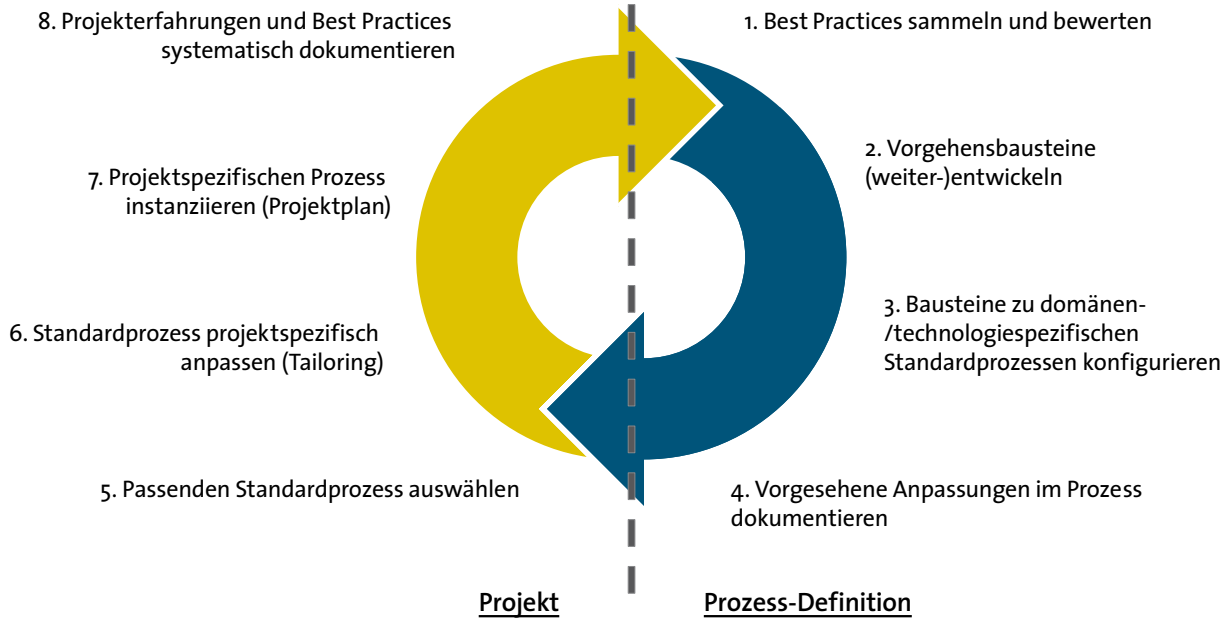


Abbildung 4: Von Best Practices zum projektspezifischen Vorgehen

1. Verbesserungsvorschläge und Best Practices werden systematisch innerhalb der Organisation erfasst und gesammelt. Hierzu dient zum einen eine für alle Mitarbeiter verfügbare Knowledge-Management-Plattform. Zum anderen existieren Issue-Tracker, die es jedem Mitarbeiter erlauben, gezielt Änderungswünsche für bestehende Vorgehensstandards zu äußern.
2. Aus den Änderungswünschen und Best Practices werden durch Prozessingenieure Vorgehensbausteine (sogenannte Process Assets) entwickelt. Hierbei handelt es sich um methodische Bausteine, Vorgaben zur Gestaltung von Ergebnissen, Checklisten etc.
3. Die Vorgehensbausteine werden zu in sich konsistenten und vollständigen Standardprozessen konfiguriert. Ein Standardprozess legt das Vorgehen für einen bestimmten Projekttyp fest. Projekttypen ergeben sich aus einer Reihe von Projektmerkmalen, wie z. B. der eingesetzten Technologie, dem erforderlichen Grad an Agilität etc. Innerhalb der T-Systems SI erfolgt die Erstellung von Standardprozessen im Wesentlichen in zwei Schritten:

   - Teile der Process Assets werden zu Methodenhandbüchern, d. h. inhaltlich vollständigen und konsistenten Methodensammlungen, zusammengestellt. Beispiele hierfür sind das Software Engineering Book (SE Book) für Individualentwicklungen, eine Variante „SAP-Book“ für SAP-Entwicklungsvorhaben etc.

- Die Inhalte der Books werden zu Standardprozessen konfiguriert. Denkbar sind hier Varianten wie ein Standardprozess für die inkrementelle Individualentwicklung oder aber ein Scrum-basiertes Vorgehen zur Individualentwicklung.

4. Parallel zur Entwicklung der Standardprozesse werden Möglichkeiten definiert, wie diese projektspezifisch angepasst (getailort) werden können. Es gilt dabei sicherzustellen, dass das Vorgehensmodell nach einem Tailoring immer noch konsistent ist und alle wesentlichen Kernelemente enthält, wie sie z. B. für eine CMMI-Zertifizierung erforderlich sind. Hierfür wird zunächst eine Reihe von Tailoring-Kriterien für Projekte definiert (z. B. Sicherheitskritikalität, erhöhte Ergonomieanforderungen etc.) Für jedes Kriterium wird durch Prozessingenieure festgelegt, welche Teile des Standardprozesses von einem Projekt ausgelassen oder geändert werden dürfen.
5. Für alle Projekte gibt es eine Reihe solcher anpassbaren Standardprozesse zur Auswahl. In Abhängigkeit vom konkreten Projektumfeld wählt ein konkretes Entwicklungsprojekt den am besten passenden Standardprozess aus.
6. Nach der Auswahl eines geeigneten Standardprozesses wird dieser projektspezifisch angepasst (Tailoring). Hierfür werden die in Punkt 4 beschriebenen Kriterien und Mechanismen genutzt.
7. Das projektspezifisch angepasste Vorgehensmodell wird zu einem konkreten Projektplan instanziiert. Ist im Vorgehensmodell beispielsweise festgelegt, dass die Entwicklung in Inkrementen erfolgen soll, so beschreibt der Projektplan konkret die Anzahl und den jeweiligen Inhalt dieser Inkremente.
8. Der Projektplan wird im laufenden Projekt umgesetzt. Die durchgängige Kette von Best Practices über einheitliche Vorgehensmodelle bis zum Projektplan stellt sicher, dass Projekte Standards einhalten und Best Practices der Organisation nutzen. Umgekehrt werden Erfahrungen und im Projekt entwickelte Best Practices systematisch erfasst, um als Eingaben für einen kontinuierlichen Verbesserungsprozess genutzt werden zu können.

3.1.4 Anforderungen an Organisation und Technik

Der hier beschriebene Ablauf wird auch innerhalb der T-Systems erst Schritt für Schritt realisiert. Für seine Realisierung müssen eine Reihe organisatorischer und technischer Anforderungen erfüllt sein.

Um Best Practices systematisch in Standards zu überführen, muss eine Organisation nicht nur dafür Sorge tragen, dass technisch die Möglichkeit besteht, Best Practices zu dokumentieren. Die neu entstehenden Best Practices müssen zudem kontinuierlich und systematisch von Experten aus unterschiedlichen Anwendungs- und Technologiedomänen auf ihre Eignung als Verbesserungen der bestehenden Vorgehensmodelle hin untersucht und bewertet werden.

Um Wiederverwendung und projektspezifische Anpassungen zu ermöglichen, müssen die Prozessstandards einer Organisation in Form eines modular aufgebauten Modells vorliegen. Es muss klare Verantwortlichkeiten für die einzelnen Bausteine geben. Für alle spezifischen Erweiterungen der Standards muss sichergestellt sein, dass Experten für deren Entwicklung und Pflege zur Verfügung stehen, die sowohl mit dem spezifischen Umfeld (wie z. B. SAP) als auch mit der bestehenden Prozesslandschaft vertraut sind.

Technisch erfordert ein hoher Grad an Wiederverwendung einerseits eine große Zahl von zu unterstützenden Projekttypen, anderseits ein maschinenlesbares Modell der Prozessstandards und eine darauf aufbauende umfassende Werkzeugunterstützung. Ein Redaktionswerkzeug zur Erstellung und Pflege unternehmenseigener Vorgehensmodelle sollte Folgendes leisten:

- Unterstützung bei der Verwaltung von Process Assets
- Möglichkeit zur Konfiguration von Standardprozessen für verschiedene Projekttypen aus bestehenden Bausteinen
- Unterstützung für das Tailoring von Standardprozessen:
  - Definieren von Tailoringkriterien und damit verbundenen Mechanismen

  - Schutz von Kernelementen vor dem Entfernen oder Verändern im Zuge des Tailorings sicherstellen
  - Tailoring im Projekt ermöglichen (d. h., Erzeugung projektspezifischer Prozessbeschreibungen, Vorlagen etc.)
- Mechanismen zur automatisierten Überprüfung der Konsistenz und Vollständigkeit der möglichen Prozessvarianten

Um Projekte in den operativen Einheiten optimal unterstützen zu können, empfiehlt sich eine Werkzeugunterstützung, die direkt auf den Prozessstandards und dessen Strukturen aufbaut. Diese sollte Projekte zunächst bei der Erstellung von standardkonformen Projektplänen auf Basis eines angepassten Standardprozesses unterstützen. Aufbauend auf einem solchen instanziierten Prozess kann dann eine umfassende, integrierte Werkzeugunterstützung zur Projektsteuerung und Durchführung etabliert werden.

3.1.5 Standardisierte Vorgehensmodelle und Frameworks an Beispielen

Im Folgenden werden drei Beispiele für die Kombination aus Vorgehensmodell und passender Toolunterstützung vorgestellt. Alle weisen sowohl prozess- als auch werkzeugseitig die Möglichkeit der projektspezifischen Anpassung auf. Je nach Projektkontext können ausgewählte Teilprozesse ergänzt oder gekürzt werden. Das Projekt bewegt sich also in einem standardisierten Rahmenwerk bestehend aus Prozess, Dokumentation und Werkzeugkasten.

Microsoft Solution Framework

Jedes Projekt ist anders. Starre Vorgehensmodelle und unflexiblen Werkzeuge der vergangenen Jahre habe diesen offensichtlichen Fakt nicht berücksichtigt. In Reinkultur eingesetzt, riefen sie schnell Kritik hervor und sorgten für eine ablehnende Haltung gegenüber standardisierten Vorgehensmodellen. Folgende Fragen wurden häufig gestellt:

- Passen die im Modell genannten Vorgehensweisen zur Unternehmenskultur? Wenn nein: Ist eine Änderung der Unternehmenskultur ohne große Verluste (Personal, Zeit etc.) überhaupt möglich?
- Kann ein Vorgehensmodell überhaupt Top-down und in Reinkultur eingeführt werden oder muss es aus dem Unternehmen heraus wachsen?
- Induzieren diese Vorgehensmodelle mit und nach ihrer Einführung nicht einen übermäßigen Verwaltungsaufwand? Ist dieser kalkulierbar? Sind Kunden bereit, dafür zu bezahlen?
- Warum beschäftigen sich die bekannten Modelle nur mit Softwareprojekten, nicht aber mit Infrastrukturprojekten?
- Welche Antworten haben die Vorgehensmodelle auf die Probleme, die sich im Lebenszyklus eines IT-Systems außerhalb von Projekten ergeben (Stichworte: Unternehmensplanung, Betrieb des Systems, strategische Systemarchitektur ...)?

Aus diesen Fragestellungen heraus hat Microsoft das Microsoft Solution Framework (MSF) entworfen. Das MSF ist eine Art Checkliste bestehend aus Gedankenstützen und Vorschlägen für Prozessmodelle, Rollen und Sichten. Das Framework ist in der Entwicklungsumgebung Visual Studio Team System integriert und kann je nach Unternehmenskultur individuell an die vorhandenen Organisations- und Ablaufschemata angepasst werden. Je nach Projektart – beispielsweise Infrastruktur- oder Entwicklungsprojekt – und dessen Umfang können die Bestandteile skaliert werden.

Zentraler Bestandteil des MSF ist das Teammodell. Darin werden sechs Rollen definiert, die den Hauptverantwortungsbereichen innerhalb eines IT-Projektes entsprechen. Die konkrete Besetzung der Rollen mit Personen sowie die möglichen Kombinationen von Rollen beschreibt das MSF getrennt von der Beschreibung der Teamrollen.

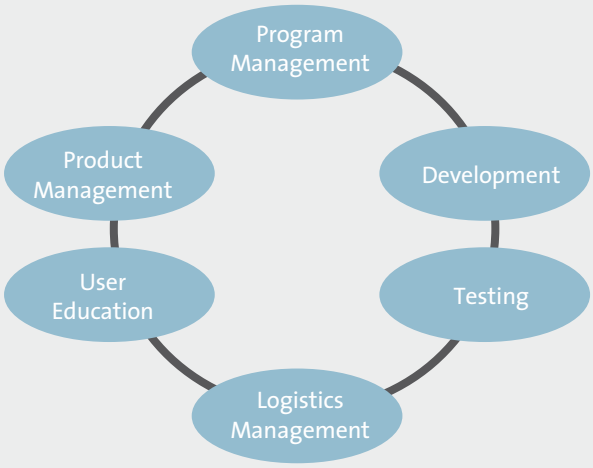


Abbildung 5: Die Definition der Teamrollen des MSF

Rolle	Haupt-Projektziele
Product Management	Zufriedener Kunde
Program Management	Lieferung innerhalb der Projektgrenzen (Budget, Zeitplan, Feature-Set)
Development	Lieferung gemäß Produktspezifikation
Testing	Release nach Lösung aller Probleme
User Education	Gesteigerte Benutzerzufriedenheit und Benutzerperformance (Leistung)
Logistics Management	Reibungslose Auslieferung und Installation

Tabelle 2: Zuordnung von Rollen zu Haupt-Projektzielen

Beim Teammodell des MSF handelt es sich um ein sogenanntes „Team of Peers"-Modell, was mit „Team gleichberechtigter Partner" nur ungenügend übersetzt werden kann. Es handelt sich nicht um ein Organigramm, das heißt: Wer wessen Vorgesetzter ist, wird nicht durch das Teammodell festgelegt. Stattdessen legt das Teammodell eine Verteilung der Verantwortlichkeit für die gemeinsame Sache (das Projekt) fest und regelt die Zusammenarbeit innerhalb eines kleinen Teams mit multidisziplinären Rollen.

Zweiter wichtiger Bestandteil des MSF ist das Prozessmodell. Es kann den Gegebenheiten im konkreten Betrieb angepasst werden. Bei der Definition eines Projekt-Meilensteins entlang der Entwicklungsprozesse steht die Übereinstimmung aller Teammitglieder (Konsens) über das Ziel im Mittelpunkt und nicht der Zeitablauf. Das MSF-Prozessmodell beschreibt also einen von Meilensteinen getriebenen Prozess. Alternativ werden mit dem Team Foundation Server auch die Prozessvorlagen für MSF Agile und MSF CMMI ausgeliefert. Nutzer können alternativ auch agile Vorgehensmodelle zu nutzen und auf Basis des Capability Maturity Model Integration (CMMI) die Prozesse kontinuierlich verbessern. Weitere Informatio-nen zum Microsoft Solution Framework finden Sie unter http://msdn.microsoft.com/de-de/library/bb979125.aspx.

Rational Unified Process (RUP)

Der Rational Unified Process (RUP) ist in vielen Branchen ein De-facto-Standard für den Softwareentwicklungsprozess geworden. Durch die Aufnahme von bewährten Praktiken (Best Practices) des Software Engineering bietet der RUP ein umfassendes Rahmenwerk zur Abwicklung unterschiedlicher Softwareentwicklungsprojekte. Im Fokus dieser Best Practices steht das Prinzip, Software iterativ und inkrementell zu entwickeln. Mit dem zyklischen Durchlaufen der beteiligten Disziplinen bietet der RUP erste Ansätze agiler Vorgehensweisen. Ein wesentliches Ziel ist es, durch schnelle Realisierung lauffähiger Software die Risiken eines Softwareprojektes zu minimieren. Iterationen werden architekturorientiert und komponentenbasiert geplant. Weitere Prinzipien sind objektorientierte und visuelle Modellierung, Verwaltung der Anforderungen, Testautomatisierung und Kontrolle der Änderungen.
Die Entkopplung der Projektphasen von den Disziplinen wird in der typischen Matrixdarstellung des RUP deutlich:

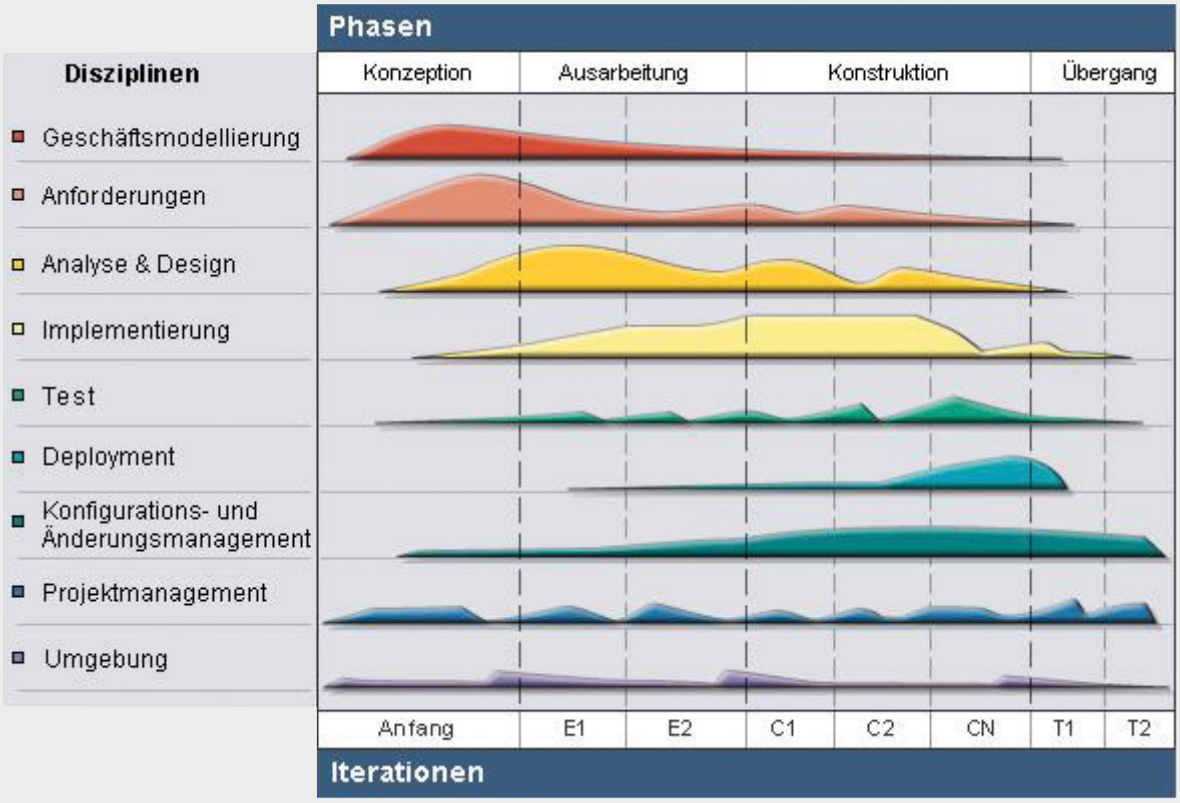


Abbildung 6: Matrixdarstellung des Rational Unified Process

Der RUP enthält detailliert ausgearbeitete „Workflows“ für alle Bereiche der Softwareentwicklung (Rollen, Aktivitäten, Produkte) sowie zahlreiche Muster (Templates) für die zu erstellenden Dokumente. Außerdem enthält der Unified Process umfangreiche Leitfäden (Guidelines) sowie Tool-Mentoren. Insgesamt besteht der RUP aus mehr als 3000 HTML-Seiten sowie gedruckter Dokumentation.

Bei dieser Vielzahl an Informationen ist es nur natürlich, dass nicht alle Definitionen für jedes Projekt passen. Eine Anpassung des Standard-RUP an Firmen- oder Projektspezifika ist also angezeigt. Die einfachste Form des Vorgehens ist wie bei einem Büfett: Man sucht sich für das jeweilige Projekt genau die passenden Prozessdefinitionen heraus, die für das Projekt notwendig und sinnvoll sind, und schreibt die Auswahl in einem Projektstandard fest.
Bei der Anpassung des RUP wird der Prozessingenieur oder der verantwortliche Projektleiter vom Rational Method Composer (RMC) unterstützt. Mit diesem Autorenwerkzeug können die Prozessdefinitionen gemäß dem zugrunde liegenden Metamodell (SPEM – Software Process Engineering Metamodel) angepasst werden. Selbst eigene Prozessdefinitionen können mit dem RMC benutzerfreundlich dokumentiert werden.
Die Wiederverwendung der Prozessdefinitionen in unterschiedlichen Kontexten wird durch die Aufteilung in Methodeninhalte und Prozessketten-Definitionen gefördert.

Beispiel für Methodeninhalt und Prozessdefinitionen:

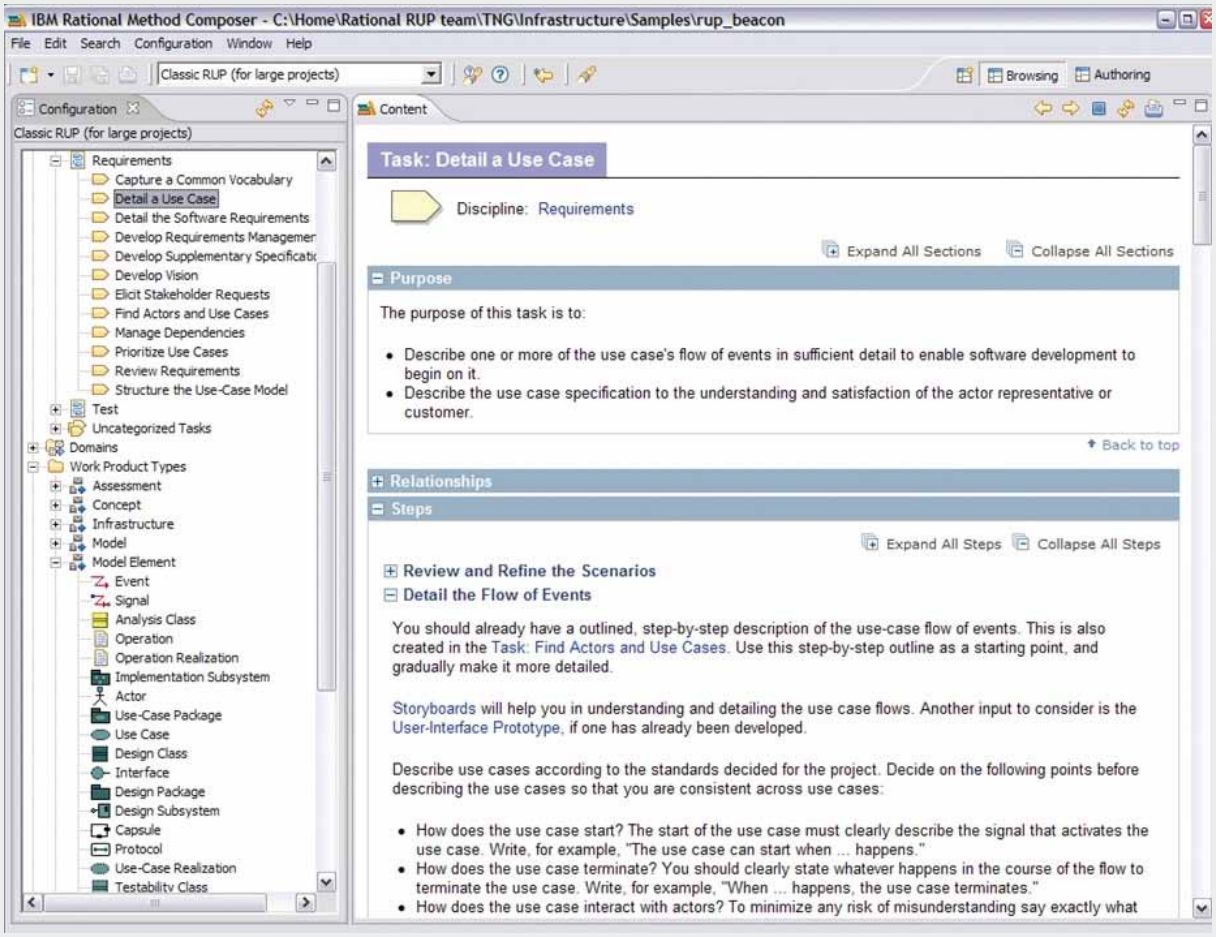
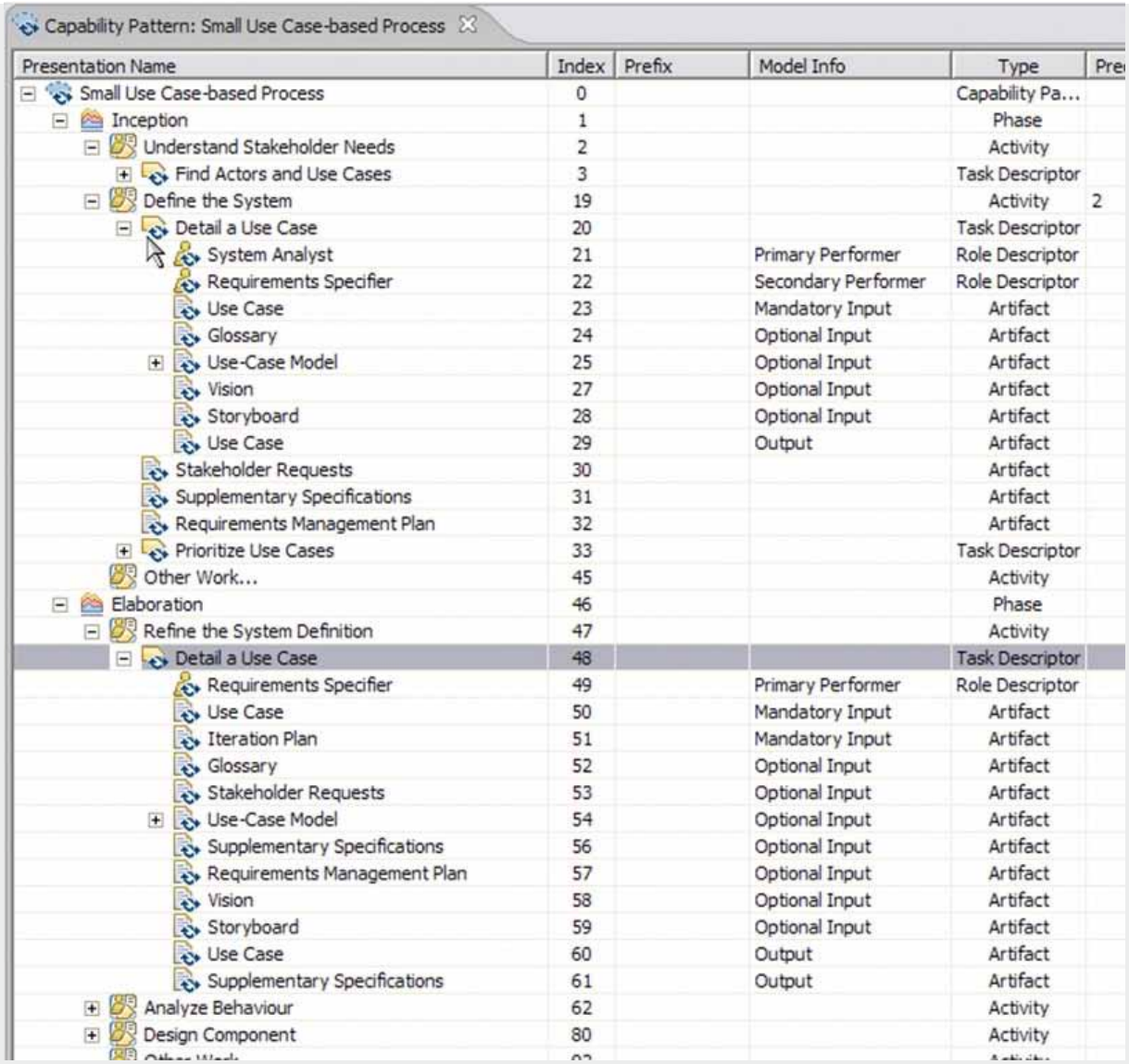


Abbildung 7: Beispiel Methodeninhalt



Capability Pattern: Small Use Case-based Process

Presentation Name	Index	Prefix	Model Info	Type	Pre
Small Use Case-based Process	0			Capability Pa...	
Inception	1			Phase	
Understand Stakeholder Needs	2			Activity	
Find Actors and Use Cases	3			Task Descriptor	
Define the System	19			Activity	2
Detail a Use Case	20			Task Descriptor	
System Analyst	21		Primary Performer	Role Descriptor	
Requirements Specifier	22		Secondary Performer	Role Descriptor	
Use Case	23		Mandatory Input	Artifact	
Glossary	24		Optional Input	Artifact	
Use-Case Model	25		Optional Input	Artifact	
Vision	27		Optional Input	Artifact	
Storyboard	28		Optional Input	Artifact	
Use Case	29		Output	Artifact	
Stakeholder Requests	30			Artifact	
Supplementary Specifications	31			Artifact	
Requirements Management Plan	32			Artifact	
Prioritize Use Cases	33			Task Descriptor	
Other Work...	45			Activity	
Elaboration	46			Phase	
Refine the System Definition	47			Activity	
Detail a Use Case	48			Task Descriptor	
Requirements Specifier	49		Primary Performer	Role Descriptor	
Use Case	50		Mandatory Input	Artifact	
Iteration Plan	51		Mandatory Input	Artifact	
Glossary	52		Optional Input	Artifact	
Stakeholder Requests	53		Optional Input	Artifact	
Use-Case Model	54		Optional Input	Artifact	
Supplementary Specifications	56		Optional Input	Artifact	
Requirements Management Plan	57		Optional Input	Artifact	
Vision	58		Optional Input	Artifact	
Storyboard	59		Optional Input	Artifact	
Use Case	60		Output	Artifact	
Supplementary Specifications	61		Output	Artifact	
Analyze Behaviour	62			Activity	
Design Component	80			Activity	

Abbildung 8:Beispiel Prozessdefinitionen

Durch diese „Modularisierung“ der Prozessinhalte ist auch die Kombination vorgefertigter Inhalte mit eigenen Vorgehensmodellen möglich. Eine Vielzahl sogenannter Methoden-Plug-ins steht zur Verfügung, z. B. für Systems Engineering, Maintenance-Projekte, DoD Architecture Framework, J2EE-Projekte, .NET-Projekte oder agile Projekte wie Scrum. Mit einem Transformationswerkzeug können z. B. Inhalte des V-Modells XT in den Method Composer importiert werden und dort mit eigenen Definitionen kombiniert werden.
Mit dem Method Composer lässt sich dann eine vollständige Dokumentation des Prozesses generieren.
Eine projektspezifische Instanziierung der Prozessdefinitionen kann mit der integrierten Entwicklungsplattform „Jazz“ vorgenommen werden. Die Prozessdefinitionen lassen sich in das Projektverwaltungstool Rational Team Concert exportieren, sodass die Definitionen für Rollen, Aktivitäten und Arbeitsergebnisse direkt für die Projektplanung herangezogen werden können (s. Kapitel 3.2.2 Heterogene Architektur und Tool-Welten).

Weitere Informationen zum RUP und zum Rational Method Composer finden Sie unter: http://www-01.ibm.com/software/awdtools/rmc/features/index.html?S_CMP=wspace
Weitere Informationen dazu s. V-Modell-Anwenderverein ANSSTAND e. V.: www.anssstand.de.

Das V-Modell XT

Das V-Modell XT [FHKS09] ist das Standardvorgehensmodell für IT-Entwicklungsprojekte im Bereich „Öffentlicher Auftraggeber". Es regelt die Zusammenarbeit zwischen (beauftragenden) Auftraggebern und (ausführenden) Auftragnehmern und legt die Aufgaben sowie die Verantwortungen fest.
Das V-Modell XT fokussiert auf die Definition von Projektergebnissen (Produkten) und abstrahiert von konkreten Abläufen. Abläufe sind orthogonal zur Ergebnismenge positioniert. Sie garantieren die Erstellung erforderlicher Produkte in definierten Strukturen und bieten dafür verschiedene mögliche Erstellungsreihenfolgen (sogenannte Projektdurchführungsstrategien) an. Durch eine Auftraggeber-/Auftragnehmer-Schnittstelle wird geregelt, welche Ergebnisse von welcher Vertragspartei zu erstellen sind und wann diese an den Partner zu übergeben sind. Dabei werden wiederum die konkreten Erstellungsprozesse abstrahiert.
Ausgehend vom V-Modell XT Referenzmodell haben verschiedene Bundesbehörden Anpassungen erstellt, welche die Bedürfnisse ihrer Organisationen erfassen (z. B. Normen und Vorgaben bzgl. des Ausschreibungs- und Vertragswesens). Beispiele hierfür sind das Bundesinnenministerium (BMI, in Zusammenarbeit mit dem Bundesverwaltungsamt) mit dem V-Modell XT Bund, die Bundesnetzagentur mit dem V-Modell XT BNetzA oder die Bundeswehr. Auf der anderen Seite ist das V-Modell XT auch für die Industrie interessant. Die EADS nahm beispielsweise eine Anpassung vor, welche die Zusammenarbeit mit öffentlichen Auftraggebern vereinfacht. Unternehmen wie z. B. die Witt-Gruppe führen ihre Systementwicklung ebenfalls auf Basis des V-Modells XT durch.
Die breite Einsetzbarkeit des V-Modells XT wird durch seinen Aufbau in Form eines Prozessframeworks inklusive Methoden und Werkzeugen zur Anpassung unterstützt [KTF10]. Ein sogenanntes generisches Referenzmodell (www.v-modell-xt.de) enthält alle Standardinhalte und Strukturen. Anpassungen können z. B. als Ergänzungen vorgenommen werden, die in das Standardmodell eingewoben werden und so eine spezialisierte, auf die jeweilige Organisation zugeschnittene V-Modell-Variante erzeugen. Den Autoren (Prozessingenieuren) steht der V-Modell XT Editor als Frontend der Werkzeuginfrastruktur zur Verfügung.
Auch im Rahmen der Projektinitialisierung stellt das V-Modell XT Unterstützung bereit. Der Projektassistent unterstützt die Projektleitung bei der projektspezifischen Anpassung, die nicht benötigte Prozessanteile entfernt. Weitere Werkzeuge wie z. B. PET [www.v-modell-xt.in.tum.de/pet.aspx] unterstützen darüber hinaus die Einbettung des V-Modells XT in verschiedene Werkzeuge für die Projektdurchführung (z. B. den Microsoft Visual Studio Team Foundation Server). Mit Werkzeugen wie Microtool in-Step [Link] wird auch Unterstützung direkt zur Projektlaufzeit angeboten. Somit ist für das V-Modell XT eine umfassende Werkzeugunterstützung – beginnend bei der Prozessdefinition bis hin zur Umsetzung im Projekt – verfügbar.
Weitere Informationen zum V-Modell XT: www.v-modell-xt.de, vmxt.blogspot.com
Weitere Informationen zu „V-Modell XT"-Werkzeugen: www.v-modell-xt.in.tum.de

3.2 Verwendung von Standards

3.2.1 Abhängigkeiten von bestehenden Standards

Wie hängt nun die Industrialisierung mit der Vielzahl existierender Standards wie SOX, Basel II oder auch CMM(I) zusammen? Wird Industrialisierung mit den oben aufgeführten Anforderungen gleichgesetzt (Modularisierung, Automatisierung etc.), so ergibt sich folgendes Bild: Jeder Standard kann als Instanz einer Industrialisierungsbestrebung aufgefasst werden. Ein Standard stellt eine mögliche Umsetzung der Industrialisierung dar. Allerdings genügt ein Standard in der Regel nicht, die Standardisierung ganzheitlich voranzutreiben. So fokussieren die vielen Prozessstandards wie ISO 15504, ISO 9001:2000, SW-CMM, CMMI oder SPICE den grundsätzlichen Aufbau der unterschiedlichen Prozesse und modularisieren und systematisieren die unterschiedlichen Geschäftsabläufe. Sie stellen damit ein mächtiges Potenzial für die Industrialisierung von Prozessen dar.

Eine weitere Klasse von Standards fokussiert das Produkt von IT-Entwicklungen, wie etwa die DoD 2167, MIL 498, ISO 15288 oder die ISO 26262. Hier werden Anforderungen an Produkte so weit standardisiert, dass sie automatisiert und wiederholbar geprüft werden können. Diese Standards stellen damit ebenfalls leistungsstarke „Werkzeuge“ der Industrialisierung von Produkten dar.

Die letzte Klasse von Standards fokussiert organisatorische und Governance-Aspekte. Standards wie PMBOK, SWEBOK, Prince2, V-Modell XT oder ISO 16085 formulieren einheitliche Anforderungen an den organisatorischen Aufbau und die jeweilig durchzuführenden Aktivitäten. Diese Standards stellen damit leistungsstarke „Werkzeuge“ für die Industrialisierung von Organisationen dar.

Wird diese Klassifikation der Standards auf die Dimensionen der Industrialisierung übertragen (s. Kapitel 2), so zeigt sich, dass Standards die Industrialisierung in allen Bereichen unterstützen. Dabei kommt es weniger darauf an, welcher Standard konkret Verwendung findet (ob die organisatorischen Strukturen z. B. entlang PMI oder Prince2 umgesetzt werden). In jedem Fall profitiert eine Industrialisierungsbestrebung davon, dass komplexe Sachverhalte (Prozesse, Produkte oder Organisationen) modularisiert werden, die Anforderungen standardisiert und bei erneutem Durchlaufen auch einer kontinuierlichen Verbesserung unterzogen werden.

Die unübersichtliche Vielzahl bestehender Standards von verschiedenen Standardisierungsgremien führt damit zu folgenden Ergebnissen bzgl. des Zusammenhangs von Standards und Industrialisierung:

- Industrialisierung kann durchaus ohne die Verwendung von Standards erfolgen. Dies ist alleine dadurch möglich, dass proprietäre Vorgaben als Standards verwendet werden. So haben viele Großunternehmen eigene Vorgaben, die zwar nicht als Standards, wohl aber als unternehmensweite Vorgaben gelebt werden.
- Industrialisierung wird durch die Anwendung von Standards unterstützt: Für den Geltungsbereich eines Standards (Prozesse, Produkte oder Organisation) unterstützt ein Standard die Industrialisierung, da Systematiken Gültigkeit bekommen, die der Zielvorgabe von Standards entsprechen (Modularisierung, Aufgabenteilung etc.).
- Industrialisierung kann trotz der Verwendung von Standards scheitern. Dies resultiert z. B. aus einer ungeschickten Kombination unterschiedlicher Standards, sodass bestimmte Bereiche völlig unberücksichtigt bleiben und sich damit der Industrialisierung, zumindest motiviert durch den Einsatz von Standards, entziehen.

3.2.2 Standards in heterogenen Architektur- und Tool-Welten

Durch die große Vielfalt existierender Infrastrukturen und Systemarchitekturen gibt es nicht „den“ Standard für Prozesse, Architekturen etc. Vielmehr sind verschiedene gleichberechtigte oder sogar konkurrierende Standards für verschiedene Bereiche der Softwareentwicklung zu

finden. Im Wesentlichen lassen sich Standards, wozu wir auch sogenannte „Quasi-Standards“ zählen, grob in den folgenden Bereichen finden:

- Plattform
- Datenaustausch und Kommunikation
- Werkzeuge

Plattformen als Standard

Die Standardisierung auf der Ebene von Plattformen wird durch einzelne Hersteller bzw. Konsortien vorangetrieben. Je nach strategischer Ausrichtung wird hierbei der Plattformbegriff unterschiedlich umrissen. Als Plattform kann beispielsweise eine Betriebssystemumgebung (z. B. die Kombination Windows Server und Client) definiert werden. In dieser Sicht wird über die zentrale Komponente Betriebssystem definiert, welche Services und Anwendungen betrieben werden können. Die Plattform definiert sich hierbei genau genommen über eine Anzahl von Basisbibliotheken und Schnittstellen, die aufbauende Services/Anwendungen nutzen können. Eine andere Sicht entsteht beispielsweise durch die Hardware. Analog zum Betriebssystem werden Funktionen zur Verfügung gestellt, die von den aufbauenden Softwarekomponenten genutzt werden können. Auch eine Mischung, also die Kombination von Hardware und Software, kann als Plattform beschrieben werden. Solche Konstruktion finden z. B. bei Apple, aber auch in vielen Bereichen eingebetteter Systeme, Anwendung. Solche Plattformen werden üblicherweise durch den Hersteller gebündelt (bundled) und nur als integriertes System vertrieben.

Auf diesen Basissystemen aufbauend werden auch Entwicklungs- und Laufzeitumgebungen oft mit dem Plattformbegriff in Zusammenhang gebracht, z. B. Java und .NET. Für solche Umgebungen gilt, was für Betriebssysteme gilt: Laufzeitumgebungen stellen Basisbibliotheken und Schnittstellen bereit, die durch aufsetzende Services/Anwendungen verwendet werden können (mehr dazu im Abschnitt Werkzeuge).

Standardisierter Datenaustausch und Kommunikation

Der größte Bedarf an Standardisierung bestand mit Sicherheit in den Bereichen Datenaustausch und Kommunikation. Daher sind hier bereits seit etlichen Jahren ausgereifte Standards verfügbar, die laufend aktualisiert und ergänzt werden. Im Bereich Kommunikation ist beispielsweise der klassische ISO/OSI-Stack zu finden, der eine Reihe etablierter Kommunikationsprotokolle wie TCP/IP, UDP und weitere ermöglicht. Diese grundlegenden Protokolle sind standardisiert, gereift und stabil. Darauf aufbauend gibt es eine Reihe weiterer Standards, wie z. B. Java Remote Method Invocation (RMI, [Quelle]), (Distributed) Component Object Model (D/COM) oder Common Object Request Broker Architecture (CORBA). Ein Teil dieser Standards ist wiederum „plattformspezifisch“, wobei hier wieder der herstellergebundene Plattformbegriff gemeint ist. Einzig RMI und CORBA sind effektiv plattformübergreifend einsetzbar, da es für sie entsprechende Laufzeitumgebungen gibt. Dass ein entsprechender Bedarf an plattformübergreifender Kommunikation besteht, ist bekannt und wurde durch Standardisierungsgremien wie die W3C [Link] durch entsprechende Standardisierungsvorhaben gewürdigt. Die Anfang der 2000er Jahre entwickelten Web-Service-Protokolle sind das Ergebnis, das breit akzeptiert wird und viele Bereiche abdeckt, die für Service-orientierte Architekturen benötigt werden (Kommunikation, Transaktion, Sicherheit, Namig etc.).

Web-Service-Protokolle ergänzen aber nicht nur die Kommunikationsprotokolle, sondern adressieren gleichzeitig auch den genormten Datenaustausch durch Nachrichten. Die Standardisierung bei Datenformaten ist ein Bereich, der stark in Bewegung ist. Für den Nachrichtenaustausch auf Basis von Web Services hat sich die Web Service Description Language (WSDL in Kombination mit SOAP) durchgesetzt, eine Sprache die auf XML basiert. XML [link zur w3c] selbst ist eine Standard-Definition für strukturierten Text. Auf diesem Standard bauen viele weitere für verschiedenste Anwendungsbereiche auf. Um einige Beispiele aufzuzählen: Business Process Modeling Language (BPML – Beschreibung von Geschäftsprozessmodellen),

XML Metadata Interchange (XMI – Datenaustausch, i. d. R. für Modelle von Entwicklungswerkzeugen) oder eXtensible Business Reporting Language (XBRL – Austauschformat aus dem Bereich Finanzen). Zunehmend findet sich XML auch in klassischen Office-Paketen wieder. Die beiden bekanntesten und konkurrierenden Formate sind hier das Open Document Format (ODF) von OpenOffice.org und Office Open XML (OOXML) im Rahmen von Microsoft Office.

Gerade die Standardisierung von Datenaustauschformaten ist eine Herausforderung, da viele Projekte durch die Vielfalt an verfügbaren Formaten an Medienbrüchen leiden. Festzustellen ist darüber hinaus, dass auch eine einfache Kopplung durch die Konvertierung von Formaten oft nicht zum gewünschten Ergebnis führt (z. B. ODF in OOMXL und zurück). Auch wenn standardisierte Formate verwendet werden, wie z. B. XMI, ist eine vollständige Interoperabilität nicht zwingend gewährleistet. XMI ist ein prägnantes Beispiel, da hier die Hersteller von Werkzeugen in der Regel Optionen zur Ausgestaltung haben und diese auch nutzen. Somit erfolgt ein Datenaustausch trotz Einsatz eines standardisierten Formats nicht immer komplikations- oder verlustfrei.

Werkzeuge

Die Standardisierung von Werkzeugen erfolgt in der Regel auf mindestens drei Ebenen:

- die Basisplattform
- das Datenformat
- organisatorische Vorgaben

Der dritte Punkt beschreibt die Standardisierung per Dekret und wirkt auf die anderen beiden Punkte, weshalb hier nicht weiter auf ihn eingegangen wird. Die Basisplattform, d. h. sowohl die Entwicklungs- als auch die Betriebsplattform, legt Standards fest, die für die Werkzeuge bindend sind (oftmals gleichzeitig auch umgekehrt). Beispielsweise ist es ohne eine Virtualisierungssoftware nicht möglich, ein Microsoft Visual Studio auf einem Mac zu betreiben. Grund hierfür sind die weiter oben (im Bereich Betriebssysteme) angesprochenen Basisbibliotheken und Schnittstellen, die für Werkzeuge erforderlich sind. Jedoch ist es auch abhängig vom Typ des Werkzeugs, ob und wie weit ein Standard greift. Für Endanwender ist es essenziell, dass die Datenformate standardisiert sind und somit die Arbeit ermöglicht wird. Für Entwickler und Rechenzentren hingegen ist die Transparenz wichtig, um Abhängigkeiten und somit potenzielle Beschränkungen zu vermeiden. Entwicklungsumgebungen wie z. B. Eclipse sind auf vielen Plattformen verfügbar und bieten ein einheitliches Arbeitsumfeld. Dies liegt daran, dass sie selbst wieder auf einer standardisierten Plattform (in diesem Fall eine Laufzeitumgebung) aufsetzen, welche die heterogene Basisplattform abstrahiert. In diesen Bereich fallen dann auch Programmiersprachen (C, C++, Java, C#) und entsprechende Bibliotheken. Diese Art der Standardisierung ist essenziell für den Architekturentwurf, da sie es ermöglicht, von Spezifika zu abstrahieren.

Für an Laufzeitumgebungen gebundene Bibliotheken gibt es zahlreiche Beispiele – zu viele, um sie hier aufzulisten. Durch die stetig wachsende Open-Source-Gemeinde entstehen oftmals „Quasi-Standards“ für einfache, aber auch komplexe Problemstellungen der Architektur. Bibliotheken und Frameworks liefern bereits vorgefertigte Subsysteme für Datenanbindung, Kommunikation, Server-Dienste, Benutzerschnittstellen und vieles mehr. Üblicherweise werden ausgewählte Bibliotheken durch Softwarehersteller bzw. Dienstleister zu Referenzarchitekturen zusammengeführt, die oftmals als Firmenstrategie gelebt werden. Wird eine solche Strategie fixiert und organisatorisch verpflichtend – sowohl für die Entwicklung als auch für den Betrieb – spricht man auch von sogenannten „Blueprints“, die detailliert beschreiben, welche Hard- und Softwareinfrastrukturen mit welchen Paketen und in welchen Versionen betrieben werden dürfen.

Standardisierung in der Architektur

Basierend auf den Plattformen definieren die Hersteller üblicherweise Standard- bzw. Referenzarchitekturen und stellen diverse Whitepapers, Definitionen und ergänzende

Bibliotheken bereit, die ihre Systeme optimal unterstützen und ansteuern. Beispielhaft sollen hier die Inhalte der „Patterns & Practices“ von Microsoft genannt werden.

Darüber hinaus widmen sich auch Standardisierungsgremien wie z. B. die Object Management Group (OMG) der Standardisierung. Prominentester OMG-Standard ist die Unified Modeling Language (UML), die eine breit akzeptierte Sprache zur Modellierung von Systemen beschreibt. Auch für Einzeldisziplinen wie z. B. das Requirements Engineering gibt es Bestrebungen, Standardisierungen auf der Ebene des Datenaustauschs vorzunehmen (z. B. Requirements Interchange Format, RIF).

Kritische Betrachtung

In vielen Bereichen gibt es Bestrebungen zur Standardisierung bzw. Normung von Elementen der Softwareentwicklung – beginnend bei Konzepten und Techniken wie z. B. der Modellierung mit UML bis hin zu integrierten Plattformen bestehend aus Hard- und Software. Die Vielfalt der Standards führt jedoch dazu, dass in vielen Bereichen oft mehrere Standards bzw. Quasi-Standards zum Teil von verschiedenen Standardisierungs- und Normierungsgremien nebeneinanderstehen. Zum Teil ergänzen sie sich; überwiegend konkurrieren sie jedoch, wie beispielsweise ein Blick auf die Datenformate ODF und OOXML zeigt.
Ein weiteres Problem ist die Interpretationsfähigkeit von Standards. Beispielhaft sei hier XMI genannt, das sich als Austauschformat für Modelle u. ä. sieht. Jeder Werkzeughersteller reichert dieses Format jedoch so an, dass ein Austausch nur bedingt möglich ist. Dasselbe Verhalten ist auch bei der Modellierung mit UML zu beobachten (deren Modelle oft im XMI-Format behandelt werden). Obwohl Darstellung und Semantik durch die OMG weitgehend festgelegt sind, ist ein Austausch der Modelle zwischen verschiedenen Werkzeugen nur bedingt möglich.

Die Standardisierung ist somit noch nicht so weit fortgeschritten wie in anderen klassischen Engineering-Disziplinen. Hier besteht in vielen Bereichen noch Handlungsbedarf.

3.3 Der Faktor Mensch

Widerstand gegen Vorgaben in der Softwareentwicklung

Im Rahmen von Softwareentwicklungsprojekten ist in der Regel auch viel Kreativität gefragt. Kreativität ist bei vielen Aufgabenstellungen der Schlüssel zum Erfolg. Darüber hinaus sind die an der Entwicklung beteiligten Personen – wie beispielsweise IT-Architekten und Entwickler – oft Individualisten, die Freiräume lieben und experimentieren wollen und müssen.

Dies steht im starken Widerspruch zu einer Standardisierung, in der Tools, aber auch Frameworks fest vorgeschrieben sind. Diese Konflikte können Entwicklungsprojekte ernsthaft gefährden oder zumindest die durch die Vereinheitlichung geplanten Synergien und Einsparungen auffressen.

In einigen Projekten ist es vorgekommen, dass Entwickler neue, von ihrer Gewohnheit abweichende Werkzeuge nutzen sollten und daher das ganze Projekt boykottierten.

Eine vergleichbare Situation entsteht bei der verpflichtenden Nutzung von Frameworks wie beispielsweise Struts oder Spring. IT-Architekten arbeiten nur ungern mit ihnen nicht vollständig bekannten Komponenten, weil sie gegebenenfalls für die Konstruktionsfehler oder Schwächen des „fremden“ Frameworks verantwortlich gemacht werden. Die Vorteile der Frameworks, Minimierung architektureller Risiken und erhebliche Einsparung von Testaufwänden, werden in dieser Situation der subjektiv gefühlten Verantwortung untergeordnet.

Um Konflikte bei Standardisierungsmaßnahmen zu minimieren, die durch dadurch entstehen, dass sich Entwickler und Architekten eingeschränkt fühlen, hat sich folgendes Vorgehen als hilfreich erwiesen:

- Transparenz für die Notwendigkeit der Veränderung (Prozess, Framework, Tool etc.) bei allen betroffenen Mitarbeiten schaffen

- bei der Auswahl bzw. dem Prozessdesign die Bedenken und Bedürfnisse der Architekten und Entwickler in den Entscheidungsprozess einfließen lassen
- den mit der Veränderung verbundenen Mehraufwand der betroffenen Mitarbeiter für Evaluation, Schulung und Eingewöhnung darstellen und als Investition in der Übergangsphase einkalkulieren

Eine weite Möglichkeit, diese oben genannten Konflikte aufzulösen, liegt darin, die Vorgaben nicht zu eng zu gestalten. Für die Umsetzung von beispielsweise Web-Anwendungen sind häufig zwei bis drei Frameworks möglich. Bei der Verwendung von bestehenden Komponenten kann ebenfalls eine Auswahl zulässig sein. Das Projektteam wählt dann nach eigenem Ermessen aus. In Unternehmen, die sehr unterschiedliche technologische Entwicklungsprojekte umsetzen, können sogenannten Green- bzw. Yellow-Lists für Frameworks und Werkzeuge geführt werden. In den Green-Lists werden Werkzeuge und Frameworks genannt, die benutzt werden müssen; in den Yellow-Lists werden diejenigen Komponenten und Tools gelistet, die verwendet werden dürfen – mit oder ohne Absprache.

Interessanterweise ist in diesem Zusammenhang zu beobachten, dass bei Ausschreibungen, in denen der Kunde Technologie, Frameworks und Tools vorgibt, die Architekten und Entwickler meist keine Einwände vorbringen und dies so akzeptieren. Von daher sollte die Schaffung einer breiten Akzeptanz für unternehmensweite Standards grundsätzlich möglich sein. Transparenz hinsichtlich Notwendigkeit und in der Entscheidungsfindung sind die Schlüssel zum Erfolg. So kann eine Standardisierung einen flexiblen Rahmen aufspannen. Die später handelnden Personen werden abgeholt und in notwendige Entscheidungen einbezogen. Schließlich darf nicht vergessen werden, dass eine industrielle Softwareentwicklung bzw. eine Software-Factory nicht bedeutet, dass dort – wie beispielsweise in Nähfabriken – am Fließband gleichförmig und monoton gearbeitet wird. Die Arbeiter in derartigen Produktionsprozessen unterscheiden sich hochgradig von denen in SW-Entwicklungsprojekten: in puncto Ausbildungsgrad und Abstraktionsfähigkeit, Qualitätsanspruch und Kreativität.

Grenzen der Flexibilität in der Softwareentwicklung

Einen anderen, bei der Industrialisierung von SW-Entwicklungsprojekten wichtigen Ansatzpunkt bilden aber die zugrunde liegenden Unterstützungsprozesse wie Konfigurationsmanagement, Anforderungsmanagement oder Testmanagement. Derartige Prozesse müssen nach Möglichkeit hochgradig standardisiert und mit einheitlichen Werkzeugen gestützt werden. Allgemein unterstützen die Projektbeteiligten eine entsprechende Standardisierung, weil das Projektrisiko für Testmanager, Produktverantwortliche oder Architekten durch unterstützende Prozesse spürbar verringert wird.

Mehr noch: Die Vereinheitlichung dieser Prozesse ist im ureigenen Interesse dieser Personen und vereinfacht – wenn zentral vorgegeben und nicht zu formalistisch – deren tägliche Arbeit, sodass sie sich auf ihre originären, teilweise kreativen Aufgaben konzentrieren können. Vorgefertigte, vorgedachte und umfassend beschriebene Prozesse müssen nicht durch das Projekt erfunden, dokumentiert und etabliert werden. Das spart Kosten und der verantwortliche Projektleiter sowie die mit der Durchführung von Unterstützungsprozessen betrauten Personen können sich auf den projektspezifischen Teil ihrer Rolle konzentrieren.

Ein wichtiger Aspekt sollte aber hier noch erwähnt werden: Mitarbeiter – egal, welche Rolle sie im Projekt einnehmen – neigen gerade bei Individualprojekten häufig dazu, standardisierte Prozesse ablehnend gegenüber zu stehen. Grund dafür sind nicht selten Erfahrungen mit überformalisierten Vorgehensmodellen. Je stärker Mitarbeiter mit sehr verschiedenen Projekttypen – beispielsweise in Bezug auf Dokumentationsaufwände und Technologien – konfrontiert werden, umso geringer ist deren Akzeptanz für Prozessstandards.

Bei Individualprojekten stößt die Standardisierung im Sinne von Industriealisierung an ihre Grenzen. Die projektspezifischen Anpassungen von Werkzeugketten und Prozessen auf Basis der Mitarbeitererfahrungen sind hier die Voraussetzung für erfolgreiche Projekte.

Falsch verstandene interne und externe Audits

Ein weiterer Punkt für vorprogrammierte menschliche Konflikte im Zuge der industriellen Softwareentwicklung sind Audits. Audits haben den Zweck, den aktuellen Qualitätsstand –beispielsweise bezüglich der Performance oder Dokumentationslage – zu prüfen. Aus nachvollziehbaren Gründen werden dafür projektexterne Personen herangezogen, um ein möglichst realistisches Bild der Situation zu bekommen. Die verantwortlichen Leiter und/oder Architekten sehen solche Audits häufig negativ. Kontroll- und Überwachungsängste mischen sich hier mit einer unterstellten Inkompetenz.

Abhilfe schafft hier, die Audits im Rahmen der Standardisierung fest vorzusehen und darüber hinaus betroffene Projektleiter und Architekten selbst mit Auditor-Funktionen zu betrauen. Zum einen verfügen diese Personen in der Regel über das notwendige fachliche Know-how, zum anderen verändert die Einnahme beider Blickwinkel die ablehnende Haltung positiv.

Zusammenfassung

Standardisierung ist eine der zentralen Dimensionen der industriellen Softwareentwicklung. Maßnahmen zur Schaffung von einheitlichen Vorgehensmodellen oder Werkzeugketten führen aber häufig zu Konflikten und Widerständen bei den betroffenen Personen. Da auf absehbare Zeit der Mensch aber den wichtigsten Wertschöpfungsbeitrag in der Softwareentwicklung durch kreatives Design und Programmierung leistet, muss gerade bei der Standardisierung mit Umsicht vorgegangen werden. Methoden aus dem Change Management und aus dem Bereich Unternehmenstransformation lassen sich hier gewinnbringen adaptieren. Standard und Werkzeugketten sollen die Architekten, Tester, Projektmanager und Programmier nicht behindern, sondern deren Effizienz und Effektivität steigern.

3.4 Fazit Standardisierung

Standardisierung ist ein integraler Bestandteil der Industrialisierung. Sie wird in der Praxis durch eine Vielzahl existierender Standards unterstützt, ohne allerdings als Garantie für eine erfolgreiche Industrialisierung dienen zu können. In vielen Bereichen der Softwareentwicklung gibt es Bestrebungen zur Standardisierung bzw. Normung. Beginnend bei Konzepten und Techniken wie z. B. der Modellierung mit UML bis hin zu integrierten Plattformen bestehend aus Hard- und Software.

Es gibt eine Vielzahl an Standards, welche dazu führen, dass in vielen Bereichen oft mehrere Standards bzw. Quasi-Standards zum Teil von verschiedenen Standardisierungs- und Normierungsgremien nebeneinanderstehen. Zum Teil ergänzen sie sich; überwiegend konkurrieren sie jedoch. Im Bereich Standardisierung herrscht noch Handlungsbedarf.

Die Standardisierung auf dem Weg einer Industrialisierung darf dabei nicht die Kreativität der involvierten Menschen behindern, sondern soll sie geschickt kanalisieren: Kreativität ist besonders dort gewünscht, wo es um unterschiedliche Lösungsideen geht, die anschließend bewertet werden. Stupide Aufgaben sowie notwendige Quality Gates auf dem Weg der Softwareerstellung werden dagegen durch restriktive Standards unterstützt, die keiner direkten Kreativität bedürfen.

Für eine erfolgreiche Industrialisierung bedeutet dies:

- Standards helfen, Industrialisierung durchzuführen.
- Industrialisierung hängt weniger von dem einzelnen verwendeten Standard ab als von einer geschickten, möglichst ganzheitlichen Orchestrierung von Standards, um Industrialisierung in Unternehmen auf breiter Front voranbringen zu können.
- Auf dem Weg zu einer Industrialisierung stellt Standardisierung eine wesentliche Vorbedingung aller weiteren folgenden Industrialisierungsschritte dar.

4 Schwerpunkt Automatisierung

4.1 Automatisierung durch geeignete Werkzeuge

Der Einsatz von geeigneten Werkzeugen ist einer der entscheidenden Faktoren, um eine systematische Unterstützung umzusetzen, wie sie von Vorgehensmodellen beschrieben wird. Werkzeuge werden in verschiedenster Form in allen wesentlichen Phasen der Entwicklung und darüber hinaus eingesetzt. Es werden hierbei verschiedene Strategien verfolgt, die von einer einfachen gemeinsamen Datenablage bis hin zu ausgefeilten integrierten Systemen reichen, die weitreichende und durchgängige Unterstützung durch verschiedene Assistenzsysteme beinhalten.

4.1.1 Traceability

Softwaresysteme sind während des Entwicklungsprozesses und der anschließenden Wartungsphase aufgrund sich ändernder Bedürfnisse und Anforderungen ständigen Veränderungen ausgesetzt. Mit dem Konzept der „Rückverfolgbarkeit“ (Traceability) können Entwicklungsschritte eines Softwaresystems nachvollzogen werden, indem die Entwicklungsartefakte der verschiedenen Herstellungsschritte über sogenannte Traceability-Links miteinander verknüpft werden.

Die Herausforderung dabei ist die Vielzahl von Artefakten, die in der Softwareentwicklung eine Rolle spielen, zu verbinden und bezüglich möglicher Auswirkungen zueinander einzuschätzen. Dies reicht von Anforderungen an die Software, Design-Modelle oder Architektur-Komponenten über den Quellcode bis hin zu produktiven Versionen von lauffähigen Softwareeinheiten. Dazu kommen noch verschiedene unterstützende Dokumentationen mit Entwurfsmustern und Vorgaben oder auch Testfälle und Testpläne. Egal, ob bei Neu- oder Weiterentwicklung von Software: Hier den Überblick zu behalten, erfordert erhebliche Disziplin und ein klar definiertes Modell, das darstellt, welche Entwicklungsartefakte wie miteinander in Relation stehen und welche Beziehungen für eine effiziente und effektive Entwicklung tatsächlich von Interesse sind.

Unter anderem verbergen sich folgende Ziele hinter dem Konzept Traceability:

- schneller Zugriff auf relevante Informationen aus benachbarten oder vorangegangenen Entwicklungsschritten
- Prüfung von Abhängigkeiten bei Änderungsbedarf
- Nachverfolgung von Änderungen und gegebenenfalls von Gründen für Entscheidungen
- Nachweisbarkeit von vollständiger Leistungserfüllung
- Nachweisbarkeit von Testabdeckung

Neben dem Traceability-Modell, das vorgibt, welche Artefakttypen sinnvoll miteinander verbunden werden sollen, steht die konkrete Verknüpfung (Trace) von den Instanzen im Vordergrund. Ein Geschäftsziel, welches mit der Software verbunden wird, führt zu funktionalen und nichtfunktionalen Anforderungen. Diese werden beim Design auf Softwaremodell-Komponenten abgebildet; ihnen müssen Testfälle zugeordnet werden. Die Modell-Komponenten werden bei der Implementierung in tatsächlich existierende Softwarekomponenten – beispielsweise Klassen – umgesetzt und anschießen in Pakete für das Deployment auf einer Laufzeitumgebung gebündelt. Alle Artefakte werden während der Entwicklung – vom Design bis hin zur Implementierung – über solche Traces direkt oder indirekt miteinander verbunden.

Je nach dem Grad, zu dem der Gesamtentwicklungsprozess automatisiert ist, und der Leistungsfähigkeit der verwendeten Tool-Kette können die Traces generiert werden oder es wird lediglich ein Hinweis auf sie bzw. eine Erinnerung für deren manuelle Erstellung erzeugt. Bei modellgetriebener Entwicklung lassen sich die Traces beispielsweise bei der Codegenerierung erzeugen und eine bilaterale Beziehung zwischen Modell-Komponente und Softwareklasse wird automatisch erzeugt. Eine bisher nicht hinreichend gelöste Herausforderung ist die automatische Modellierung von Abhängigkeiten zwischen Anforderungen untereinander oder zwischen Anforderungen und Testszenarien bzw. konkreten Testfällen. Hier lassen sich im praktischen Einsatz nur Platzhalter automatisch erzeugen. Beispielsweise werden zu jeder

Detailanforderung aus dem Anforderungsmanagement-Werkzeug im Testmanagement entsprechende Links hinterlegt. Gepaart mit beispielsweise der quantitativen Regel –„jeder Anforderung muss durch mindestens drei Testfälle und zwei Testszenarien beim Integrationstest geprüft werden!“ – geben die Links einen ersten Anhaltspunkt für die Tester bei der Erstellung von Testfällen.

Wichtig für die Traceability über den kompletten Software-Lebenszyklus hinweg ist die Interoperabilität der Werkzeugkette. Aktuell gibt es für die unterschiedlichen Entwicklungsdisziplinen Tools, die mehr oder weniger interoperabel sind, und deren Schnittstellen untereinander begrenzt sind. In einem konkreten Fall stellten sich z. B. die Fragen: Lassen sich Anforderungen aus dem Requirements-Management-Tool mit Design-Komponenten aus meinem UML-Werkzeug verknüpfen? Oder wie einfach kann der Tester bei einer Änderung eines Testfalls nachschlagen, welche verschiedenen Anforderungen ursprünglich einmal zu dieser Teststellung geführt haben?

Eine Lösung scheint im Konzept der integrierten Entwicklungsumgebungen bzw. stark integrierten Werkzeugketten – dem sogenannten Suite-Ansatz – zu liegen, weil die Interoperabilität zwischen den Werkzeugen verschiedener Anbieter noch nicht den Kundenbedürfnissen entspricht. Alternativ werden Informationen redundant in den verschiedenen Tools gehalten und regelmäßig abgeglichen. Es gibt jedoch Entwicklungsszenarien, in denen Spezialwerkzeuge die Anforderungen besser erfüllen als ihre Konkurrenz-Module in den Tool-Suiten.

Die Fragen rund um Traceability und Interoperabilität enden häufig in der Grundsatzdiskussion bezüglich Werkzeugketten: Best of Breed vs. Best of Suite. Eine Entscheidung zwischen passgenauem Werkzeug inklusive Integrationsprojekt oder Suite-Ansatz mit möglicherweise nicht exakt den benötigten Funktionen ist notwendig. Der folgende Abschnitt wird die Vor- und Nachteile der beiden Toolstrategien vorstellen. Ungeachtet dessen ist die Traceability ein wichtiges Konzept im Rahmen der ingenieurmäßigen Softwareentwicklung und damit auch Bestandteil des Industrialisierungsansatzes.

4.1.2 Die Toolstrategien „Best of Breed“ und „Best of Suite”

Ebenso umfangreich wie die verschiedenen Heran- und Vorgehensweisen in der Entwicklung sind die Werkzeuge, die den Entwicklungsprozess unterstützen. In vielen einzelnen Disziplinen sind Werkzeuge verfügbar, die Unterstützung verschiedener Aufgabenbereiche anbieten. Oftmals sind solche Werkzeuge direkt aus der Notwendigkeit entstanden, spezifische Tätigkeiten und Methoden zu unterstützen. Beispielhaft sei das Unit Testing genannt, das sich als Testmethode großer Beliebtheit erfreut und in vielfältiger Form von verschiedenen hoch spezialisierten Tools unterstützt wird. Viele Unternehmen und Behörden haben bisher aus diesem Pool verfügbarer Tools (oftmals aus pragmatischen Gründen) einzelne ausgewählt und danach versucht, diese zu durchgängigen Werkzeugketten zu verbinden. Diese Strategie wird als „Best of Breed“ bezeichnet und führt dazu, dass gute Tools für den jeweiligen Prozessschritt, wie z. B. Anforderungsmanagement oder Testen, genutzt werden. Die Integration der Tools gestaltet sich jedoch sehr aufwendig und erschwert eine ganzheitliche Sicht auf Projekte, insbesondere dann, wenn für einen Prozessschritt verschiedene Tools parallel verwendet werden. Bei der Integration der einzelnen Spezialwerkzeuge kommt es immer wieder zu Medienbrüchen, welche die Kommunikation, die Interaktion und den Datenaustausch in Projekten erschweren. Bereits einfache Fragestellungen wie: „Wie wird die Verbindung von Testsuiten und QS-Plan hergestellt?“ münden in erheblichem organisatorischen Aufwand, sofern eine Integration der verwendeten Werkzeuge nicht von Haus aus gegeben ist.

Bei der „Best of Suite“-Strategie wird versucht, eine lose gekoppelte Werkzeugkette durch eine einheitliche Plattform zu ersetzen. In der Regel wird zuerst eine integrierte „Application Lifecycle Mangement (ALM)“-Plattform gewählt und dann im nächsten Schritt geprüft, welche Zusatztools noch notwendig sind, um nicht abgedeckte Prozessschritte ebenfalls zu unterstützen. Diese Zusatztools müssen sich dann über standardisierte Schnittstellen standardisiert in diese Plattform integrieren lassen bzw. müssen entsprechend angepasst werden. Projektgruppen verlagern dadurch den Integrationsaufwand zu

den Toolherstellern und können ein sehr umfangreiches Reporting nutzen, weil alle Informationen in einem zentralen Repository abgelegt werden. Werden die verschiedenen Rollen im „Best of Breed"-Ansatz durch jeweils individuelle Werkzeuge bedient, findet sich im „Best of Suite"-Ansatz eine integrierte Projektinfrastruktur, die durch Generierung verschiedener Sichten die Projektdaten für die jeweiligen Rollen filtern kann. Idealerweise werden die Daten so aufbereitet, dass sie in rollenspezifische Spezialwerkzeuge importiert werden können, sodass auch die Zusammenarbeit im Projektteam auf der Grundlage einer gemeinsamen Datenbasis erfolgen kann. Dabei wird stets ein synchroner und konsistenter Datenbestand angestrebt, auf dessen Grundlage ein Reporting etabliert werden kann.Der Aufbau von Werkzeugumgebungen unterscheidet sich je nach dem gewählten Grundansatz. „Best of Breed" geht selektiv vor und versucht, eine möglichst schlanke, an den Bedürfnissen des Projektteams orientierte Auswahl zu treffen (vgl. Tabelle 3, in Anlehnung an [KKD09]).

Im Rahmen des „Best of Suite"-Ansatzes wird auf einem Basispaket eine Infrastruktur aufgebaut, indem fehlende Bestandteile identifiziert und gegebenenfalls ergänzt werden. In der Regel sind die „Best of Suite"-Pakete jedoch schon weitgehend vorkonfiguriert. Am Beispiel des Team Foundation Server (vgl. Tabelle 4, in Anlehnung an [KKD09]) ist dies gut zu erkennen.

Durch die Fertigung einer Suite durch einen Hersteller ist der Integrationsgrad natürlich wesentlich höher als bei einer Zusammenstellung von Tools unterschiedlicher Hersteller, was die Risiken der Medienbrüche zwischen einzelnen Entwicklungsschritten und Disziplinen vermindert. Jedoch ist auch die Herstellerbindung bei der Auswahl der Werkzeuge zu berücksichtigen. Offene Infrastrukturen nach dem „Best of Breed"-Ansatz weisen eine höhere Flexibilität auf, leiden jedoch unter der oftmals mangelnden Integration.

Tool	Disziplin	Beschreibung
Linux Server	-	Backend für Dienste wie Datenbanken, Mail etc.
Subversion	Configuration Management	zentrale Datenablage
Open Office	-	Infrastruktur für Textverarbeitung, Kalkulationen etc.
Eclipse	Entwicklung/ Design etc.	Framework für die Entwicklung, den Entwurf, Zugriff auf SCM-Systeme etc.
BugZilla oder Mantis	Entwicklung (Support)	Unterstützung für Bug- oder Issue-Tracking

Tabelle 3: Beispielhafte „Best of Breed"-Infrastruktur (Auszug)

Tool	Disziplin	Beschreibung
Windows Server	-	Backend für Dienste wie Datenbanken, Mail etc.
SharePoint	Collaboration & Communication	erweiterte webbasierte Infrastruktur für Kommunikation und Kollaboration (integriert in TFS oder als Add-on)
SQL Server	Configuration Management, Business Intelligences (Reporting)	Datenbankdienste für Ablage von Entwicklungsartefakten und allg. Dateien sowie Reporting (integriert in TFS oder als Add-on)
Office	-	Infrastruktur für Textverarbeitung, Kalkulationen etc. (integriert in TFS)
Visual Studio	Development	Entwicklungsumgebung
Team Foundation Server	Development Collaboration & Communication	Basisinfrastruktur für Teamarbeitsumgebungen, die verschiedene Einzelapplikationen (SharePoint, Office etc.) zusammenführt

Tabelle 4: „Best of Suite"-Infrastruktur am Beispiel TFS (Auszug)

Best of Suite am Beispiel Microsoft Visual Studio Team System

Das Microsoft Visual Studio Team System ist ein Beispiel für eine integrierte Toolplattform gemäß der „Best of Suite"-Strategie. Das Team System besteht aus folgenden Komponenten:

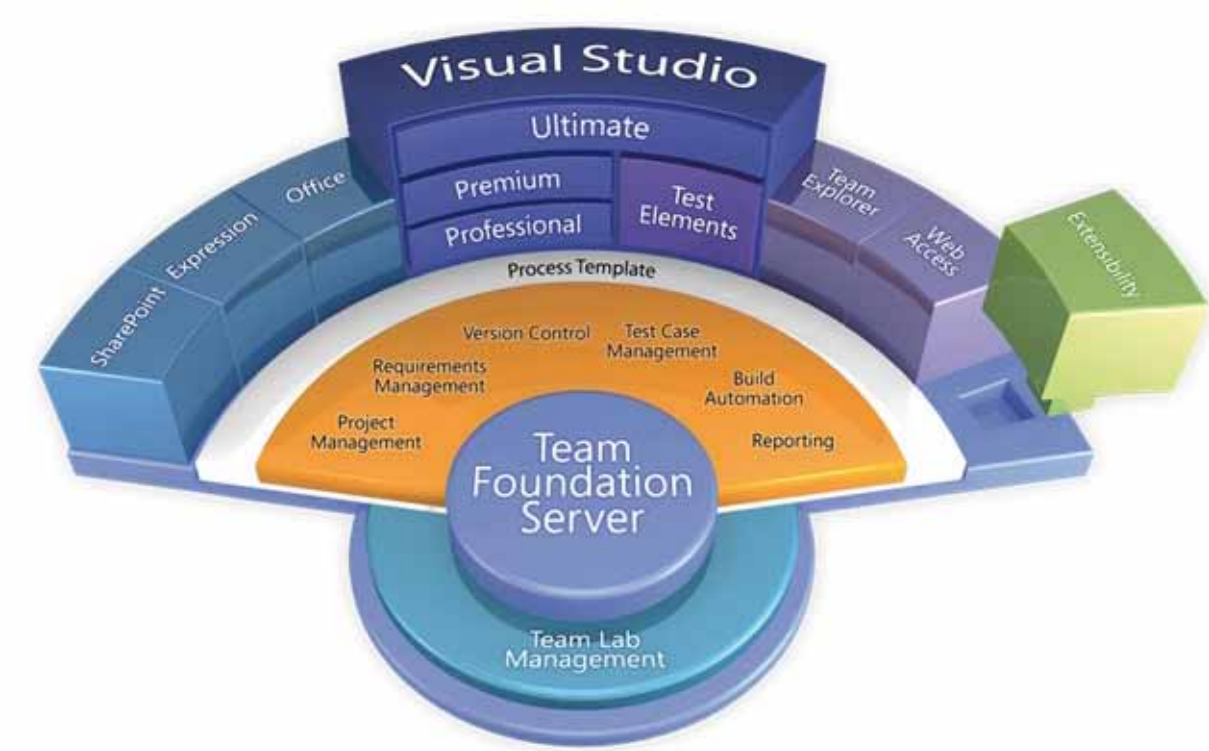


Abbildung 9: Visual Studio 2010

Visual Studio als Client ist in der Regel bekannt und wird daher nicht näher erläutert. Als zentrales Repository dient der Team Foundation Server. Dort werden strukturierte Daten wie z. B. User Stories, Tasks, Test und Bugs sowie unstrukturierte Daten wie z. B. Quellcode und Dokumente erfasst. Umfangreiche Berichte, die den Projektstatus bezüglich Implementierungs- und Testabdeckung zeigen, können Out-of-the-Box genutzt werden. Das Process Template ist für jedes Projekt frei wählbar und kann auch bei Bedarf angepasst werden. Weitere Informationen finden Sie unter http://www.microsoft.com/germany/visualstudio/.

Best of Suite am Beispiel Jazz und Rational Team Concert

Jazz ist die IBM-Technologieplattform zur Softwareentwicklung in Teams.
Die Jazz-Plattform basiert auf den Open-Web- und OSGi-Alliance-Standards und stellt eine ausbaufähige Architektur zur Verfügung, die das Ziel verfolgt, die Prozesse der Softwarebereitstellung in Zukunft noch kooperativer, produktiver und transparenter zu gestalten. Das Produkt ist in hohem Maße auf die Anforderungen globaler Teams abgestimmt und vereint Awareness-Funktionen für Personen, Projekte und Prozesse mit automatisierten Funktionen, die den Softwarelebenszyklus beschleunigen und die Projektgovernance verbessern. Die Jazz-Plattform baut auf der Eclipse-Technologie auf und dient als Basis für die IBM Rational Software Delivery Platform und die Erweiterungen der Partner.

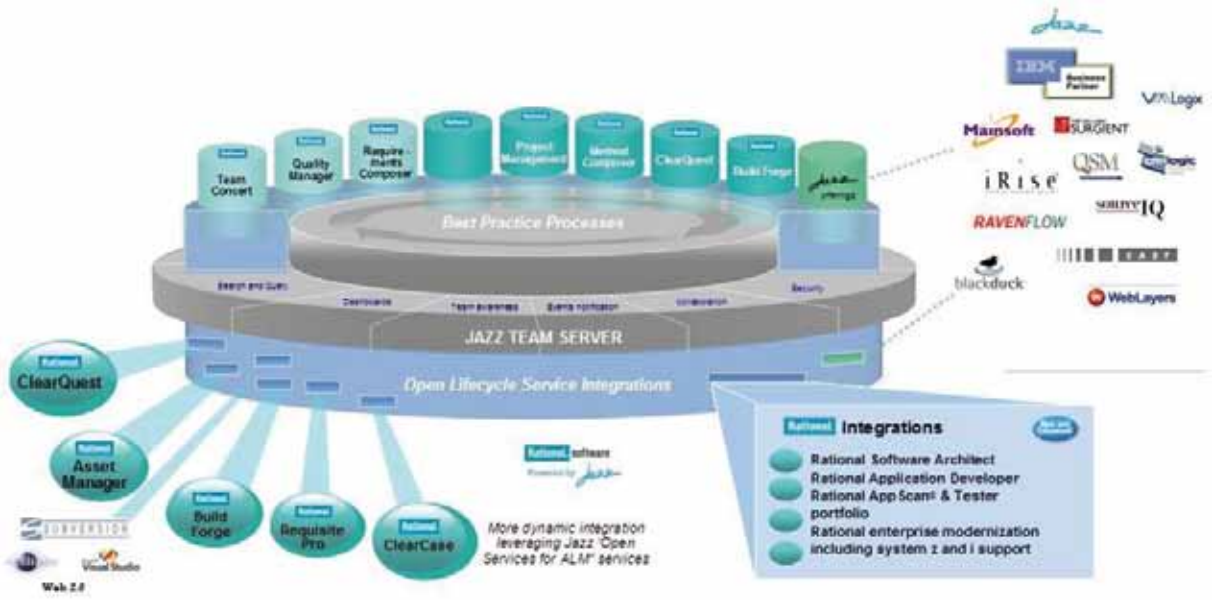


Abbildung 10: Aufbau Jazz-Plattform

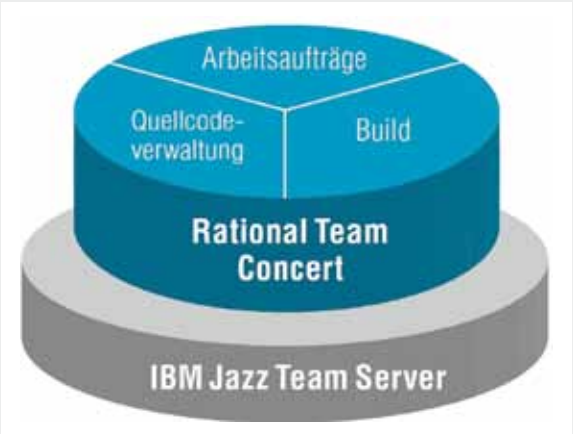


Abbildung 11: Rational Team Concert

Auf der Technologieplattform Jazz basiert Rational Team Concert. Rational Team Concert vereint einen Web-2.0-Ansatz zur Teamzusammenarbeit mit integrierter Aufgaben-, Build- und Softwarekonfigurationsverwaltung. Unter Verwendung konfigurierbarer Portalansichten können die Teammitglieder auf Projektinformationen zugreifen, wie z. B. News und Ereignisse oder aktueller Buildstatus. Sie können so erkennen, woran gerade gearbeitet wird oder welche Änderungswünsche vorliegen. Sie können außerdem sehen, woran die Kollegen gerade arbeiten und wer gerade online und erreichbar ist.
Rational Team Concert erlaubt die Automatisierung von Prozessen einschließlich Benutzerunterstützung und ermöglicht damit, das erforderliche Maß an Prozessvorgaben zu definieren, um vorhersehbare Ergebnisse zu erreichen. Und da nicht alle Projekte und Organisationen gleich sind, bietet

Rational Team Concert die Möglichkeit, Prozessregeln zu konfigurieren und damit individuelle Anforderungen zu erfüllen. Man kann mit einem der mitgelieferten vordefinierten Prozessmodelle beginnen und dann bei Bedarf jederzeit eigene Regeln definieren und optimieren.
Die Softwareentwicklung im Team gleicht dem Zusammenspiel von Musikern in einer Band. In beiden Fällen geht es um ein ausgewogenes Verhältnis zwischen den Fähigkeiten der Einzelnen und dem Zusammenwirken als Team. Die IBM „Rational Team Concert"-Software stellt eine Entwicklungsumgebung zur Verfügung, die Einzelpersonen und Teams dabei hilft, qualitativ hochwertige Arbeitsergebnisse zu erzielen. Das Produkt bietet eine grundlegende Softwareversionierung, Arbeitspaket- und Arbeitsbereichsverwaltung und die Unterstützung paralleler Entwicklung. Außerdem wurde Rational Team Concert dafür ausgelegt, geografisch verteilte Entwicklungsteams zu verbinden, die Produktivität jedes Einzelnen sowie des Teams zu steigern, die Entwicklungszyklen zu verkürzen und qualitativ hochwertige Software in kürzester Zeit bereitzustellen.
Wenn Sie weitere Informationen zu Jazz oder Rational Team Concert wünschen oder an dem kommerziellen, aber offen durchgeführten Entwicklungsprojekt zum Aufbau der Jazz-Plattform teilnehmen möchten, besuchen Sie folgende Website: www.jazz.net.

Der „Best of Breed"-Ansatz am Beispiel T-Systems

Die T-Systems hat vor vier Jahren begonnen, ihren Softwareentwicklungsprozess zu standardisieren. Dabei wurde von Beginn an der Prozess unabhängig von den Werkzeugen definiert. In einem parallelen Projekt wurden die Softwareentwicklungswerkzeuge evaluiert und anschießend für den definierten Prozess standardisiert. Die folgenden Punkte waren bei der Entwicklung des Werkzeugkastens entscheidend:
Es kamen nur Werkzeuge in die Auswahl, für die bereits positive Erfahrungen in der T-Systems existierten. Gute Erfahrungen mit dem Werkzeug und geschulte Mitarbeiter waren bei der Evaluation ein zentrales Kriterium.
Die Werkzeuge mussten den Vorgaben des Prozesses folgen, d. h., es wurde bewusst nach spezialisierten Werkzeugen für einen Prozessschritt gesucht.
Die Anzahl der Werkzeuge pro Prozessschritt sollte minimal sein, d. h., im besten Fall deckt ein Werkzeug alle Aspekte des Prozessschrittes ab. Um unterschiedliche Typen von Projekten zu unterstützen, wurden für ausgewählte Prozessschritte mehrere Werkzeuge in den Werkzeugkasten aufgenommen. So ist neben Eclipse auch VisualStudio im Werkzeugkasten, da diese Produkte (meist) unterschiedliche Sprachen unterstützen. Es ist aber neben Eclipse keine weitere Java-IDE (Integrated Development Environment) gesetzt.
Alle Werkzeuge müssen zentral betrieben und administriert werden können, dabei gleichzeitig auch mandantenfähig sein. Es muss sichergestellt sein, dass viele Projekte auf derselben Infrastruktur betrieben werden können, ohne dass Daten aus einem Projekt von nicht autorisierten Benutzern eingesehen werden können.
Die mit der so zusammengestellten Werkzeugkette gemachten Erfahrungen waren überwiegend positiv. So konnten durch wenige Administratoren hunderte von Projekten parallel unterstützt werden. Jedes Projekt hatte dabei nur die Kosten zu tragen, die es auch wirklich verursacht hat.
Ein wichtiger, nicht zu unterschätzender Aspekt ist die Integration der ausgewählten Werkzeuge. Die Reife der Werkzeuge und die von ihnen unterstützten Standards zum Datenaustausch stellen mitunter eine Herausforderung dar. Auf der anderen Seite hat man durch eigene Integration von verschiedenen Werkzeugen mehr Freiheiten, für unterschiedliche Projektbedürfnisse die Toolunterstützung des Entwicklungsprozesses flexibel anzupassen.

4.2 Domänenspezifische Sprachen und modellgetriebene Softwareentwicklung

Ziel der modellgetriebenen Softwareentwicklung ist es, die Softwareentwicklung auf ein höheres Abstraktionsniveau zu heben und so die Lücke zwischen der semantischen Problemlösung und der Implementierung in einer bestimmten Technologie zu schließen. Sobald die Lösung auf abstrakter Ebene definiert ist, werden Zug um Zug zuvor vernachlässigte Details hinzugefügt, bis eine ausführbare Software verfügbar ist. Diese Lücke wird in der Literatur auch als „abstraction gap" bezeichnet.

Die Abstraktionsebene zu heben, wurde bereits in den achtziger Jahren mit Aufkommen der CASE-Tools (Computer Aided Software Engineering) versucht. Sie ermutigten Entwickler dazu, ihre Software grafisch darzustellen, beispielsweise mithilfe von Zustandsmaschinen, Struktur- oder Datenflussdiagrammen [Schmo6]. Aus den entstandenen Modellen wurde anschließend ausführbarer Programmcode generiert. Allerdings waren die grafischen Modelle zu generisch, um eine kontextfreie Beschreibung der gewünschten Software zu liefern und passten nicht immer optimal zur gewünschten Programmiersprache. Das Ergebnis war oft hochkomplexer Quellcode, der nach der Generierung noch manuell angepasst werden musste. Die ursprünglichen Modelle waren nach diesen Änderungen oft veraltet, da die gängigen CASE-Tools Änderungen im Quellcode nur schwer erkennen, geschweige denn grafisch abbilden konnten.

Um die genannten Probleme zu lösen, kombiniert die modellgetriebene Softwareentwicklung zwei wichtige Ansätze: domänenspezifische Sprachen (engl.: Domain Specific Languages, DSLs) und Modelltransformatoren und -generatoren [Schmo6].

Domänenspezifische Sprachen

Eine domänenspezifische Sprache (DSL) modelliert die Grundkonzepte eines bestimmten fachlichen Gebiets wie beispielsweise Onlinedienste im Finanzsektor, E-Commerce-Anwendungen, CRM-Systeme oder andere klar abgegrenzte Fachbereiche. Die Charakteristika eines Fachgebiets werden in Metamodellen dargestellt, welche präzise die jeweiligen Entitäten und deren Semantik beschreiben [Schmo6].

"A modelling language is a visual type system for specifying model-based programs. It raises the level of abstraction, bringing the implementation closer to the vocabulary understood by subject matter experts, domain experts, engineers and end-users." [GrSho4]

Eines der erfolgreichsten Beispiele einer domänenspezifischen Sprache findet sich in WYSIWIG (What you see is what you get)-Editoren für grafische Benutzeroberflächen. Während in den Anfängen der Softwareentwicklung grafische Benutzeroberflächen nur von hoch qualifizierten Programmierern erstellt werden konnten, erlauben aktuelle Editoren und Wizzards nahezu jedermann, leistungsstarke Oberflächen zu entwickeln. Möglich wurde das durch die Definition einer in den GUI-Editoren implementierten, hoch spezialisierten Sprache. Sie beschreibt exakt die verschiedenen Elemente und deren Beziehungen und Einschränkungen zueinander (beispielsweise können Schaltflächen nur innerhalb eines Fensters vorkommen). Weitere bekannte Beispiele sind ereignisgetriebene Prozessketten (Event Driven Process Chains, EPC) oder das Entity-Relationship-Modell [Belt+07]. Mithilfe von domänenspezifischen Sprachen sollte es also möglich sein, ein Online-Shopping-System mit automatischer Bonitätsabfrage zu erstellen, ohne sich über die Kompatibilität Sorgen machen zu müssen. Zusammengefasst bieten DSLs die folgenden Vorteile [Belt+07:]

- Spezifikationen können mithilfe von DSLs schneller und exakter erstellt werden.
- Änderungsanforderungen können präziser und unmissverständlich beschrieben werden.
- Spezifikationen sind kontextfrei und lassen keinen Raum für Interpretationen.
- Codegeneratoren können für eine bestimmte Domäne entwickelt werden und sind daher weitaus effizienter als frühere CASE-Tools.
- Die Umwandlung von Modellen in Quellcode durch einen Codegenerator ist weniger Fehleranfällig als die manuelle Implementierung eines Produkts.

Transformation Engines und Generatoren

Sobald ein Softwaresystem in einem klar abgegrenzten Problemraum spezifiziert wurde, können die mithilfe der domänenspezifischen Sprache erstellten Modelle entweder in ausführbaren Code oder in noch genauere Modelle umgewandelt werden. Letztere sind insbesondere vor dem Hintergrund eines großen „abstraction gap" hilfreich, beispielsweise wenn ein Programm auf verschiedenen Plattformen laufen soll. Diese sogenannten Zwischenmodelle fügen der Programmlogik dann die Besonderheiten der jeweiligen Plattform hinzu. Um jedoch solche Transformationen zu ermöglichen, müssen die Generatoren über Metamodelle des Ursprungs- und Zielzustands sowie Übersetzungsregeln verfügen. Während die Modelle üblicherweise im Rahmen von DSLs vorliegen, müssen die Transformationsregeln entsprechend definiert werden [Pham+07].

In ihrem Buch „Software Factories" identifizieren Greenfield und Short die modellgetriebene Softwareentwicklung als eine der Kerninnovationen für industrielle Softwareentwicklung. Sie differenzieren zwischen vertikalen oder horizontalen Transformationen. Vertikale Transformationen bereiten ein Modell auf detaillierterer Ebene auf oder führen es direkt in Quellcode über. Ein Beispiel wäre hier die Definition, über welche Web Services eine bestimmte Programmlogik abgebildet werden soll. Horizontale Transformationen hingegen fügen zusätzlich benötigte Aspekte wie beispielsweise Sicherheitsfeatures, aber auch die Unterstützung einer bestimmten Produktlinie hinzu.

4.3 Testautomatisierung

Das Testen als Teil des Softwareentwicklungsprozesses unterliegt dem Anspruch steigender Produktivität und ständiger Optimierung. Durch zunehmende Größe und Komplexität von Systemen und die Nachweispflicht für deren Qualität ändern sich die Anforderungen an das Testmanagement. Die Einhaltung von Zeit- und Kostenvorgaben ist auch bei diesem Teil des Entwicklungsprozesses ein wesentlicher Aspekt. Redundante Abläufe sollen vermieden, Synergieeffekte zwischen den einzelnen Testaktivitäten genutzt und Softwaretests beschleunigt werden. Gleichzeitig sollen die Qualität der Tests selber und die Testabdeckung möglichst hoch sein. Auch nichtfunktionale Anforderungen an die Softwaresysteme wie beispielsweise Security oder Performance sollen geprüft werden.

Die Automatisierung der Tests im Sinne der industriellen Softwareentwicklung hilft, Softwareentwicklung insgesamt effektiver und zuverlässiger zu gestalten.

4.3.1 Nutzen der Testautomatisierung

Automatisiertes Testen ist die programmgesteuerte Ausführung der unterschiedlichen Testaktivitäten wie die Identifikation und Erstellung von Testfällen (Testspezifikationen, Testdaten und -abläufe), Testdurchführung, Auswertung und Beurteilung der Testergebnisse und die Dokumentation der Abläufe und Ergebnisse.

Im Rahmen der genannten Aktivitäten gibt es verschiedene Möglichkeiten der Automatisierung. Die Palette reicht von Testfallgenerierung auf Basis von Systemmodellen über automatische Testdatengenerierung bis zur automatischen Testausführung und Auswertung.
Die möglichen Vorteile der Testautomatisierung sind vielfältig, jedoch sind auch bei dieser Art der Automatisierung zunächst die notwendigen Rahmenbedingungen zu schaffen und Vorarbeiten zu leisten.

Vorteile und Nutzenpotenziale:

- Wiederverwendung der Testfälle – Einmal implementierte Testfälle können immer wieder durchlaufen werden. Darüber hinaus werden Testfälle immer wieder exakt gleich durchlaufen und so ist es ausgeschlossen, dass ein gefundener Fehler durch den veränderten Ablauf eines Testfalls hervorgerufen worden ist.
- Reproduktion von Fehlern – Im Rahmen manueller Tests können entdeckte Fehler manchmal nicht reproduziert werden. Mittels Dokumentation und Protokollen ermöglicht das automatisierte Testen

das Nachvollziehen der entdeckten Fehler und ihrer Entstehung.

- Einmalige Eingabe von Testdaten – Testdaten müssen nur einmal generiert und eingegeben werden.
- Dokumentation und Nachweisführung – Durch die Dokumentation und die ständige Wiederholung von Tests wird die Qualität der Software, für welche vorher ein Maß definiert werden muss, messbar. Die Dokumentation ermöglicht es, Fehlerkorrekturen und Entwicklungsfortschritte intern nachzuvollziehen oder auch gegenüber dem Kunden darzulegen.
- Vergleich von Testergebnissen nach Änderungen an der Software – Aufgrund neuer Anforderungen oder zur Fehlerbehebung werden Änderungen an der Software vorgenommen, die bei manuellem Testen oftmals nicht den Aufwand rechtfertigen, alle Testfälle erneut zu durchlaufen, bevor die Weiterentwicklung fortgesetzt wird. Allerdings wäre dies notwendig, um das Auftreten neuer Fehler, insbesondere durch unerwünschte Nebeneffekte, auszuschließen bzw. sicherzustellen, dass ein Fehler tatsächlich behoben worden ist. Bei manuellen Tests werden häufig bereits erfolgreich getestete Fälle nicht wiederholt getestet, obwohl durch unvorhersehbare Nebeneffekte neue Fehler auftreten können, die so übersehen werden. Testautomatisierung hilft, einen schnellen Überblick zu erhalten und verringert die Hemmschwelle, alle Testszenarios erneut komplett durchzuführen.
- Simulierung von nichtfunktionalen Anforderungen – Für das Durchführen von Last- und Performanztests ist automatisiertes Testen unverzichtbar, da eine manuelle Durchführung nicht möglich oder absolut unwirtschaftlich wäre. Darüber hinaus bieten Testwerkzeuge die Möglichkeit, verschiedene Umgebungen (bspw. Betriebssysteme, Hardwarekomponenten) und Schnittstellen zu simulieren. Dies verringert den Testaufwand erheblich.
- Ausblenden der Fehlerkomponente Mensch – Fehler, die beim manuellen Testen herausgefunden werden, müssen ihre Ursache nicht in der Software haben, sondern können auch die Folge eines Bedienfehlers oder sonstigen Versagens der testenden Person sein. So können das unvollständige Ausführen von Testfällen oder das unbeabsichtigte Verändern der Testumgebungsbedingungen zum Auftreten von Fehlern führen.
- Höhere Testabdeckung – Die werkzeugunterstützte Durchführung von Softwaretests bietet die Möglichkeit einer höheren Testabdeckung, weil durch Automatisierung viel mehr Testfälle durchlaufen werden können als bei manuellem Testen.
- Testen von Modellen und damit Testen in frühen Entwicklungsphasen – In einigen Bereichen der Entwicklung hat sich das Model-driven Software Development (MDSD) durchgesetzt. Unter günstigen Bedingungen können die dem zukünftigen System zugrunde liegenden Modelle selbst getestet werden. Darüber hinaus kann aus dem Systemmodell ein Testmodell abgeleitet werden. Bei diesem Ansatz ist aber besondere Vorsicht geboten. Wenn der Quellcode und die Testfälle aus dem gleichen Modell generiert werden, besteht die Gefahr, in Wirklichkeit nur das Codegenerierungswerkzeug zu testen.

Das Testen steht nicht wie manchmal angenommen am Ende der Softwareentwicklung. Vielmehr sollte das Testen als begleitender Prozess eines Softwareprojekts verstanden werden, welcher von Beginn an integriert wird. Der Softwareentwicklungsprozess unterliegt durch Tests einer ständigen Kontrolle und Fehler können frühzeitig erkannt und beseitigt werden. Ständige Tests sind nur mit Automatisierung durchführbar.

4.3.2 Herausforderungen bei automatisiertem Testen

Programmgesteuertes Testen bietet viele Möglichkeiten, effizienter und effektiver zu arbeiten. Damit dies möglich wird, müssen jedoch einige Punkte beachtet werden. Denn auch wenn ein Großteil der Testaktivitäten werkzeugunterstützt durchgeführt werden kann, ist eine vollständige Automatisierung des Testprozesses kaum möglich und eine sorgfältige und qualifizierte Begleitung des Testprozesses bleibt unabdingbar.

- Sorgfältige Planung und Testmanagement – Unabhängig davon, ob manuell oder automatisiert getestet wird, sollten die Testaktivitäten bereits zu Beginn

eines Softwareprojekts geplant und vorbereitet werden. Ein dediziertes Testmanagement und somit auch die organisatorische Verankerung innerhalb eines Projekts sind wichtig für den Erfolg der Testaktivitäten. Das eigentliche Testen darf nicht zugunsten der Testautomatisierung vernachlässigt werden. Falsch implementierte Testfälle durchlaufen die Testroutine und liefern eventuell Ergebnisse, deren Aussagekraft fragwürdig ist. Zu berücksichtigen ist dabei auch der richtige Mix unterschiedlicher Automatisierungswerkzeuge. Es ist zu entscheiden, welche Funktionalitäten auf Unittest-Ebene und welche Funktionalitäten auf Systemtest-Ebene über die Oberfläche getestet werden sollen.

- Erwartungen prüfen: Grenzen der Testautomatisierung – Oftmals werden zu hohe Erwartungen an die Testautomatisierung gestellt. Werkzeuge (tools) versprechen sorgenfreie und reibungslose Abläufe und der Begriff „Automatisierung“ suggeriert die Abgabe der Arbeit an ein Programm. Das ist zum Teil richtig: Testwerkzeuge haben eher unterstützenden Charakter und eine vollständige Automatisierung aller Testaktivitäten ist mittelfristig unrealistisch. So ist zum Beispiel die automatisierte Generierung von Testfällen nur auf Basis eines Testmodells möglich. Dieses muss wiederum selbst qualitätsgeprüft sein, um die gewünschte Testabdeckung zu erreichen.
- Die automatische Erstellung von Testspezifikationen ist ebenfalls nicht trivial. Oftmals kommt man nicht umhin, Testfälle manuell zu erstellen. Noch dazu müssen für automatisierte Tests die Spezifikationen in maschinenlesbarer Form vorliegen und operationalisierbar sein, was in der Regel mehr Zeit in Anspruch nimmt als die Formulierung von Testspezifikationen für manuelles Testen. Der Aufwand für die Erstellung der Spezifikation für automatisiertes Testen kann je nach Komplexität des Systems um den Faktor 5 bis zu 20 höher liegen. Damit liegt nahe, dass sich der Ansatz nur bei Produktlinien betriebswirtschaftlich rechtfertigen lässt.
- Durch Menschen verursachte Fehler können daher auch durch Testautomatisierung nicht vollständig ausgeschlossen werden. Durch automatisierte Tests werden nicht unbedingt mehr Fehler gefunden als beim manuellen Testen. Auch der Gesamtaufwand verringert sich nicht immer, weil die Vorbereitung der Automatisierung häufig unterschätzt wird. Die Erfahrungen der vergangenen Jahre haben gezeigt, dass gerade bei evolutionären Produktlinien insbesondere die automatische Testdurchführung auf Basis von schrittweise entwickelten Testfällen sinnvoll ist und durch eine Vielzahl von Werkzeugen unterstützt wird.
- Es kann ein Engpass bei der Testauswertung entstehen, wenn bei der automatisierten Testdurchführung viele Soll-Ist-Abweichungen entstehen, die manuell zu begutachten sind. Hier helfen beispielsweise Werkzeuge zum Vergleich und zum Filtern von Testergebnissen, um nur relevante Abweichungen interpretieren zu müssen.
- Zu beachten ist auch der nicht zu unterschätzende Anpassungsaufwand der automatisierten Testfälle, wenn sich das zu testende Programm geändert hat.
- Kosten-Nutzen-Analyse ist notwendig – Testautomatisierung gibt es nicht umsonst: Passende Werkzeuge, Training und sorgfältige Implementierung sind Voraussetzung. Die Investition in den Aufbau großer Testfall-Bibliotheken und umfangreicher Testdaten macht nur dann einen Sinn, wenn deren Verwendung nicht nur auf ein Projekt beschränkt bleibt.
- Werkzeugauswahl und Integration in den Entwicklungsprozess – Die Palette der am Markt angebotenen Testwerkzeuge ist im Funktionsumfang sehr unterschiedlich und hinsichtlich der verschiedenen Entwicklungs-Frameworks und des Testprozesses nicht immer einfach zu integrieren. Darüber hinaus eignen sich nicht alle Werkzeuge für alle Technologien und bei einem Technologiewechsel entsteht unter Umständen Anpassungsaufwand.

4.3.3 Zusammenfassung zur Testautomation

Richtig eingesetzt kann Testautomatisierung gerade im Bereich der Testdurchführung erheblich Kosten senken. Allerdings muss berücksichtigt werden, dass die Testergebnisse und deren Aussagekraft erheblich von den Testdaten, den Testfällen und der Testabdeckung abhängen.

In diesem Bereich sind die größten Anfangsinvestitionen verborgen, weil nur die sorgfältige Vorbereitung der Automatisierung im Rahmen des Testmanagements und die Pflege der Testfälle und Daten nachhaltig Erfolg verspricht. Testautomatisierung kann man nicht einfach kaufen: Das Zusammenspiel von Werkzeugen, Testmanagement und auch der qualifizierten Mitarbeiter sind der Schlüssel zum Erfolg.

4.4. Fazit Automatisierung

In bereits stark industrialisierten Domänen spielt Automatisierung eine zentrale Rolle bei Effizienz- und Effektivitätssteigerungen. Jedoch werden die Automatisierungsansätze vorzugsweise im Bereich der Produktion und nicht im Entwicklungs- und Design-Umfeld eingesetzt. Die Erstellung von Software hat ihren Schwerpunkt aber genau in diesem Bereich.

Aufgrund des komplexen Prozesses der Softwareentwicklung sind unterstützende Werkzeuge unabdingbar, um einzelne Entwicklungsschritte möglichst weit zu automatisieren. Durchgängige Werkzeugketten helfen dabei, Informationsverlust zu verhindern und die Abhängigkeiten zwischen den Artefakten der Softwareentwicklung kontrollieren zu können. Ob ein „Best of Breed“- oder ein „Best of Suite“-Ansatz bei der Werkzeugauswahl mehr Vorteile bringt, hängt vom bevorzugten Prozessmodell, dessen gewünschter Flexibilität und der Nutzung modellgetriebener Verfahren ab. Für Letztere stehen aber noch nicht für alle Anwendungsgebiete geeignete DSLs und Modelle zur Verfügung, weil deren Erstellung sehr komplex und kostenintensiv ist. Sind die Modelle inklusive DSL und passender Modelltransformationen einmal erstellt bzw. stehen zur Verfügung, lassen sich mit MDSD deutliche Verbesserungen wie präzise, standardisierte und unmissverständliche Beschreibungen von Anforderungen bis hin zu weniger fehleranfälligem Quellcode durch Codegenerierung erreichen. Eine vergleichbare Situation ist im Bereich Testautomatisierung zu erkennen. Im Bereich der Testdurchführung lassen sich bei verbesserter Qualität Aufwände senken, wenn Testdaten, Testfälle, ein geeignetes Testmanagement sowie passende Werkzeuge zur Verfügung stehen.

5 Schwerpunkt Wiederverwendung

Wiederverwendung findet in der IT-Welt auf unterschiedlichen Ebenen und in verschiedenen Phasen des Softwareentwicklungsprozesses statt. Es kann zwischen technischer, konzeptioneller und organisatorischer Wiederverwendung unterschieden werden.

Technische Wiederverwendung:

Das am weitesten verbreitete Beispiel für technische Wiederverwendung in der Softwareentwicklung ist die Nutzung von Standard- bzw. Programmier-Bibliotheken. Hier kann bereits vorhandene feingranulare Funktionalität erneut genutzt werden.

Die Nutzung von Frameworks gilt auch als technische Wiederverwendung. Der wesentliche Unterscheid zur Wiederverwendung von Bibliotheken ist, dass Frameworks bereits ein umfangreiches Set an Funktionalitäten besitzen und die Bestandteile des Frameworks auf die entwickelten Ergänzungen zugreifen.

Die nächste Ebene ist Wiederverwendung von komplexen Softwarekomponenten, welche über einen definierten fachlichen oder technischen Funktionsumfang verfügen. Auf die Funktionen der Komponenten wird über Schnittstellen, beispielsweise in Form von Service-Aufrufen, zugegriffen. Diese Variante verbirgt sich hinter der Idee komponentenbasierter Softwareentwicklung. Insbesondere bei Produktlinien, in denen große Teile der Funktionalität über mehrere Versionen oder Produktvarianten unverändert bleiben, kann ein hoher Grad der Komponentenwiederverwendung erreicht werden (s. Abschnitt 6.2 „Software-Produktlinien“).

Konzeptionelle Wiederverwendung:

Wiederverwendung von Konzepten in frühen Phasen der funktionalen Spezifikation oder des technischen Entwurfs ist deutlich einfacher als die Wiederverwendung von Software. Einfache Beispiele dafür sind Analyse- und Software-Patterns, komplexere Beispiele sind fachliche und technische Referenzarchitekturen. Darunter sind wiederverwendbare fachliche und technische Konzepte wie beispielsweise Kommunikationsmuster oder Architekturmodelle zu verstehen, die sich auf unterschiedliche Anwendungsbereiche, Kundenbedarfe oder Branchenlösungen übertragen lassen. Die Umsetzungen der jeweils beschriebenen Konzepte finden individuell und abhängig von der konkret verwendeten Technologie statt. Der Grad der Wiederverwendung steigt hier in dem Maße, in dem der Anwendungsbereich im Vorfeld bereits fachlich oder technisch eingeschränkt oder spezialisiert wurde. Das heißt, die Spezialisierung (s. Kapitel 6) unterstützt den Einsatz von Wiederverwendung. Das trifft auch in besonderem Maße auf die Wiederverwendung von Software zu.

Organisatorische Wiederverwendung:

Von Wiederverwendung auf organisatorischer Ebene kann gesprochen werden, wenn bei der Produkterstellung regelmäßig auf vordefinierte (unternehmensspezifische) Prozess- und Tool-Standards, auf erprobte Prozessabschnitte und Methoden oder auf eingespielte Entwicklungsteams für bestimmte Aufgabenstellungen zurückgegriffen wird. Hierzu zählt auch die Nutzung von Erfahrungen und Best Practices. Eine besondere Ausprägung dieser organisatorischen Widerverwendung stellen die Software Factories dar (s. Abschnitt 6.4).

Erfolgreiche Wiederverwendung ist letztendlich immer eine Frage der organisatorischen Umsetzung und der Steuerung. Wiederverwendbare „Bausteine“ zur Verfügung zu stellen, ist meist deutlich einfacher, als eine Organisation zu befähigen, diese effektiv und effizient zu nutzen.

Folgende Hindernisse verringern den Grad der Wiederverwendung in der Softwareentwicklung erheblich:

- mangelnde Unterstützung durch das Management und fehlende Motivation
- keine oder unvollständige Dokumentation der verfügbaren Komponenten

- fehlendes Vertrauen in die bestehenden Komponenten
- fehlende systematische Unterstützung der Software-Wiederverwendung

5.1 Komponentenbasierte Softwareentwicklung

Die Idee, Software in klar abgrenzbare Einheiten zu unterteilen und daraus die gerade benötigte Anwendung zusammenzubauen, ist wahrscheinlich so alt wie die Softwareentwicklung selbst. Wissenschaftlich erwähnt wurde sie erstmals 1968 auf der NATO Software Engineering Conference. Schon damals wurde ein eigenes Marktsegment vorgeschlagen, das sich insbesondere mit wiederverwendbaren Komponenten, gruppiert nach Präzision, Fehleranfälligkeit, Allgemeingültigkeit und Performance, beschäftigt [Mcil69, p136]. Eine genauere Definition des Komponentenbegriffs findet sich beispielsweise in „Component Software“ von Clemens Szyperski [Szyp99, p.27]:

“A software component is a unit of composition with contractually specified interfaces and context dependencies only. A software component can be deployed independently and is subject to composition by third parties”.

Eine Komponente benötigt demnach ein abgegrenztes Umfeld, mit dem sie über klar definierte Schnittstellen kommuniziert, ohne jedoch ihre Implementierung offenzulegen. Idealerweise ist eine solche Komponente sprach- und plattformunabhängig und erlaubt die Zusammensetzung neuer Komponenten oder ganzer Anwendungen.

In vier Schritten zur komponentenorientierten Anwendung

Qualification: Bestehende Komponenten werden identifiziert und auf ihre Eignung im gegebenen Kontext untersucht. Das Ergebnis dieser Qualifikation bestimmt, ob die gewünschte Funktionalität neu entwickelt werden muss oder aus bestehenden Artefakten gewonnen werden kann. Dabei können verschiedenste funktionale und nichtfunktionale Anforderungen betrachtet werden, wie beispielsweise Algorithmen oder Schnittstellen sowie Fehlertoleranz oder Performance.

Adaptation: Falls erforderlich, werden die geeigneten Komponenten in einem nächsten Schritt an das konkrete Entwicklungsvorhaben angepasst. Solche Anpassungen können z. B. Wrapper für andere Plattformen oder auch die Integration zusätzlicher Funktionalität sein. Hierbei kommt es jedoch darauf an, ob die Komponente als White-, Grey- oder Blackbox implementiert ist: Erstere liegt im Quellcode vor und erlaubt umfangreiche Änderungen, jedoch auf Kosten von Kompatibilität und Austauschbarkeit. Blackbox-Komponenten liegen üblicherweise binär vor und lassen nur geringfügige, äußere Änderungen zu. Die so erzielte Stabilität geht jedoch auf Kosten der Flexibilität. Ein Mittelweg findet sich in Greybox-Komponenten. Sie erlauben zwar keine Änderungen am Quellcode, bieten jedoch umfangreiche Application Programming Interfaces (APIs) zur Anpassung und Erweiterung.

Assembly: Die so qualifizierten und angepassten Komponenten werden in einem dritten Schritt zu neuen Komponenten oder Applikationen innerhalb eines Frameworks zusammengesetzt. Das Framework stellt dabei das eingangs erwähnte Umfeld der Komponente dar, und bietet grundlegende infrastrukturelle Dienste, wie beispielsweise Kommunikation oder Hardwaremanagement.

Maintenance & Enhancement: Nachdem eine komponentenorientiert entwickelte Anwendung in Betrieb gegangen ist, beginnen Wartung und Weiterentwicklung der eingesetzten Komponenten. Wird bei einem Kunden beispielsweise ein Softwarefehler festgestellt und behoben, kann die korrigierte Komponente in allen weiteren Anwendungen ausgetauscht werden. Ebenso sind Weiterentwicklungen mit verbesserter Performance oder Funktionalität denkbar.

Aktuelle Technologien und Frameworks

Zur Implementierung solcher Anwendungen haben sich inzwischen verschiedene Technologien und Frameworks auf dem Markt etabliert. Die derzeit wohl bekanntesten

und am verbreitesten sind das CORBA Component Model, JAVA EE/EJB, COM+ sowie .NET, die im Folgenden kurz vorgestellt werden.

CORBA Component Model: Das CORBA Component Model (CCM) basiert auf der Common Object Request Broker Architecture (CORBA) und erweitert diese um Merkmale und Dienste zur Entwicklung komponentenorientierter Systeme. Es besteht aus den folgenden vier Hauptbestandteilen: Das Component Model beschreibt den grundsätzlichen Aufbau einer Komponente, ihre Eigenschaften, Schnittstellen und Datenformate, Referenzierung sowie Konfiguration. Die Component Implementation Definition Language (CIDL) stellt eine Art Bauplan einer komponentenorientierten Anwendung dar, indem sie das Zusammenspiel, die Struktur sowie Zustände von Komponenten anhand deren Component Model definiert. Letzteres verfügt über ein Component Implementation Framework (CIF), in dem die programmierspezifischen Rahmenbedingungen und -funktionalitäten einer Komponente festgelegt sind. Ausgeführt werden sie in einer Laufzeitumgebung, die über das Container Programming Model definiert und in beispielsweise einem Applikationsserver implementiert ist [Andro4, pp.273-274]. Je nach Verfügbarkeit der Laufzeitumgebungen ist das CORBA CCM damit plattform- und sprachenunabhängig.

JAVA EE/EJB: Eine weitere Architektur zur Entwicklung komponentenbasierter Anwendungen stellt die Enterprise-Java-Beans (EJB)-Architektur dar. Dabei wird die Geschäftslogik in EJBs implementiert, die innerhalb sogenannter EJB Container ausgeführt werden. Die Container stellen den EJBs die erforderliche Infrastruktur wie beispielsweise Transaktionshandling, Sicherheitsdienste oder die Steuerung konkurrierender Zugriffe zur Verfügung. Die EJB Container wiederum werden auf EJB-Servern ausgeführt, welche die Container von den Besonderheiten des jeweiligen Betriebssystems abschirmen. Da EJBs keinerlei Benutzerschnittstellen bieten, müssen sie über Client-Anwendungen und entsprechende EJB-Konnektoren angesprochen werden. Dabei können Web Services, IIOP (Corba Internet Inter ORB Protocol), natives Java oder der Java Message Service genutzt werden. Trotz der Plattformunabhängigkeit von Java bzw. der EJB-Architektur werden keine sprachenunabhängigen Datenformate angeboten.

COM+: Das von Microsoft entwickelte Component Object Model (COM) ist eine Architektur zur Entwicklung komponentenorientierter Anwendungen auf Basis eines Windows Betriebssystems. Komponenten können in Visual Basic, C++ oder C# implementiert werden, verfügen aber über eine Interface Description Language zur sprachenunabhängigen Kommunikation mit anderer Software. Die Kommunikation einer Anwendung mit COM+-Komponenten erfolgt dabei über einen Service Control Manager (SCM), der die jeweilige Komponente lädt, erforderliche Services wie beispielsweise Transaktionshandling bereitstellt und die Anfragen weiterleitet. Die Verbindung zu Komponenten in anderen Prozessen oder auf verteilten Systemen wird über eine Referenz auf ein transparentes Proxy-Objekt hergestellt, das mit dem realen Objekt über verschiedene Kommunikationsmechanismen verbunden sein kann. [Greu03, pp.278–280]

.NET: Eine weitere Möglichkeit komponentenorientierte Stand-alone-Anwendungen zu realisieren, stellt Microsofts .NET-Framework dar. Mithilfe der Common Language Runtime (CLR) werden alle Laufzeit-Aspekte einer Anwendung oder Komponente verwaltet, unabhängig von der Programmiersprache in der diese erstellt wurde. Neben diesem Laufzeitsystem und diversen Klassenbibliotheken bietet .NET verschiedene Subsysteme zur Entwicklung bestimmter Lösungen an. Im Bereich der komponentenorientierten Softwareentwicklung sind dies insbesondere der BizTalk Server sowie die Unterstützung von Web Services. Zusätzlich können .NET-Anwendungen auch mit Komponenten und Services des COM+-Modells interagieren. Der BizTalk Server vereint unterschiedliche Standards für den Informationsaustausch. Er bedient sich dabei der Orchestration von Komponenten und Services, einem XML-basierten Messaging Service sowie der Unterstützung verschiedener Übertragungsprotokolle wie beispielsweise HTTP/S, SOAP oder SMTP. Zudem können in .NET Web Services zur Abbildung komponentenorientierter Anwendungen genutzt werden. Sie implementieren eine bestimmte Funktionalität in einer beliebigen Sprache und auf einer beliebigen Plattform. Eine übergreifende

Interaktion wird durch standardisierte Übertragungsprotokolle und Datenschnittstellen erreicht.

5.2 Fazit Wiederverwendung

Systematische technische, konzeptionelle und organisatorische Wiederverwendung basiert auf Wissensmanagement und lernenden Organisationen. Dafür sind Anfangsinvestitionen für die Werkzeug-Bereitstellung, Prozessdefinition und unternehmensweite Regeln – beispielsweise zur Dokumentation – notwendig. Darüber hinaus sind auch entsprechende Ressourcen in jedem Projekt zu planen, welche neue Komponenten für den gemeinsamen Komponentenpool entwickeln.

Wiederverwendung von Softwarekomponenten oder Programmierbibliotheken sowie die Nutzung von Frameworks im Sinne der Industrialisierung führen zu höherer Produktivität und zu verkürzten Entwicklungszeiten.

Darüber hinaus lassen sich Kosten und in der Regel auch die Qualität besser abschätzen. Häufig werden bei der Softwareerstellung auch Konzepte wie beispielsweise Design- oder Architektur-Patterns verwendet. Die große Herausforderung auf allen Ebenen der Wiederverwendung ist der Aufbau der Bibliotheken und die fortlaufende Verwaltung der Funktionen und Komponenten. Hier fallen Anfangsinvestitionen für Werkzeug-Bereitstellung, Prozessdefinition und die Formulierung unternehmensweiter Regeln sowie für die Bereitstellung entsprechender Ressourcen in jedem Projekt zur Erweiterung des Wiederverwendungspools an.

Ein weiterer aktueller Trend zur Wiederverwendung von Funktionen sind Service-orientierte Architekturen. Ihr Ziel ist die flexible Wiederverwendung von Services auf fachlicher und technischer Ebene. Die Service Repositories stellen dabei eine Bibliothek von zur Laufzeit verfügbaren Funktionen dar. Mehr dazu finden Sie im BITKOM-Leitfaden SOA unter www.soa-know-how.de.

6 Schwerpunkt Spezialisierung

6.1 Entwicklungslinien und Disziplinen

- Wo vor zehn Jahren die Kenntnis mehrerer Programmiersprachen und ein gutes analytisches Verständnis ausgereicht haben, um in unterschiedlichsten Projekten mit jeweils anderen Technologien unterschiedliche Rollen wahrzunehmen, ist dies – mit einem professionellen Anspruch – heute so kaum noch möglich. Gründe dafür sind primär
- die deutlich erhöhte Komplexität von (gewachsenen) IT-Systemen und den abzubildenden Geschäftsprozessen
- die Heterogenität der heutigen Systemlandschaften und die Vielfalt der Umgebungen
- die hohe Anzahl an nutzbaren Klassen des jeweiligen Frameworks, an Werkzeugen und 3rd-Party-Komponenten (kommerziell und Open Source)

Die Vorstellung, dass heute ein Java-Entwickler in der Lage wäre, nach kurzer Einweisung professionell Cobol zu entwickeln, ist ebenso falsch wie die Vorstellung, dass er einen hochkomplexen und ausgefeilten Test-Prozess per se managen oder unterstützen kann.

Der Übergang vom Generalisten zum Spezialisten beschreibt diesen Wandel gut.

Daher ist Spezialisierung auf einzelne Technologien und Disziplinen hilfreich, um beispielsweise Organisationen und Teams für eine professionelle Softwareentwicklung entsprechend aufzustellen und weiterzuentwickeln. Das folgende Leitbild soll dabei diese Spezialisierung nach Technologien aufgrund ihrer Besonderheiten veranschaulichen und eine feinere Strukturierung bzw. Eingruppierung ermöglichen:

Unter einer „Entwicklungslinie“ werden alle Techniken, Methoden, Werkzeuge, Komponenten und Produkte verstanden, die hier zum Einsatz kommen. Die in der Grafik gezeigten Entwicklungslinien sind nahezu disjunkt:

Während in der Entwicklungsline „Java/J2EE“ beispielsweise Werkzeuge, Komponenten und Produkte wie Eclipse, Swing, JBoss, JUnit, SQL/JDBC, Hibernate, Subversion oder Bugzilla eine große Rolle spielen, sind viele von ihnen im Microsoft-Umfeld praktisch ohne Bedeutung.

Oft sind innerhalb einer Entwicklungslinie bei vielen Mitarbeitern mehrere Fähigkeiten (Disziplinen) wie z. B. Design, Entwicklung, Test gut ausgeprägt.

Eine „Disziplin“ ist zunächst die Fokussierung auf eine oder mehrere Rollen, die im Zuge der Softwareentwicklung relevant sind:

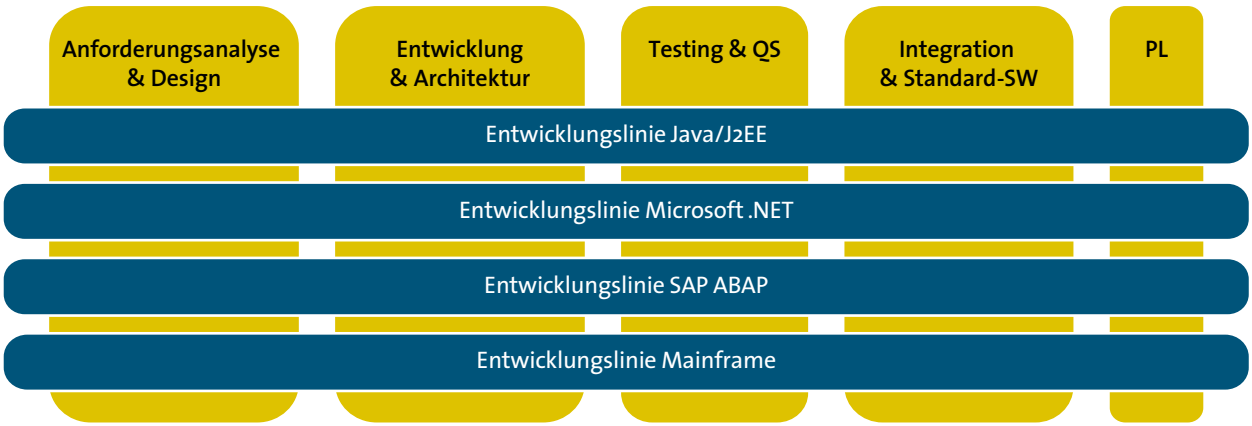


Abbildung 12: Spezialisierung nach Entwicklungslinien und Disziplinen

So stehen beispielsweise in der Disziplin „Test & QS“ Aktivitäten wie die Erstellung eines Test-Konzeptes und von Testfällen, die Verwaltung selbiger – unter Zuhilfenahme einschlägiger, i. d. R. technologieunabhängiger Werkzeuge – oder die Definition griffiger Abnahmekriterien im Vordergrund. Hierfür sind entsprechende Ausbildungen und Fähigkeiten notwendig, die durch Zertifikate wie das ISTQB dokumentiert werden können.

In der Disziplin „Entwicklung & Architektur“ sind Aktivitäten wie das Entwerfen von Architekturen mit entsprechenden Werkzeugen (bspw. Rational Rose oder Visio) sowie die die eigentliche Programmierung (d. h., das Umsetzen von Design in Programm) angesiedelt. Umfangreiche Kenntnisse über den Softwareentwicklungsprozess (d. h. Konfigurations-Management, Nutzung einer Versionskontrolle) und einschlägiger Methoden (bspw. IBM Rational Unified Process© oder SCRUM) kommen dazu.

Anders als bei den Entwicklungslinien spannen sich heute typischerweise bei vielen Mitarbeitern innerhalb einer Disziplin die Kenntnisse nicht über mehrere Entwicklungslinien hinweg:

Ein SAP-Projektleiter wird nur schwer ein Java-Projekt leiten können. Und wer Anforderungsanalyse und Design im Host-Umfeld beherrscht, kann dies nicht zwangsläufig auch für eine Microsoft .NET-Anwendung (zumindest zeigt das die Erfahrung heute).

Ob und inwieweit eine Spezialisierung von Mitarbeitern in nur ein einzelnes Segment (d. h. den Schnittpunkt Entwicklungsline und Disziplin) hinein sinnvoll ist, ist individuell zu betrachten.

Wie Entwicklungslinien und Disziplinen tatsächlich in einer Organisationsform zusammengefasst sind und wie weit dies sinnvollerweise umgesetzt werden sollte, hängt natürlich sehr stark ab von

- dem Kunden bzw. dem Dienstleistungsportfolio,
- der Größe der Organisation und
- den vorhandenen Mitarbeitern.

In jedem Fall empfiehlt es sich, für Organisationsformen im Umfeld der Softwareentwicklung eine Trennung gemäß dem obigen Gedankenmodell vorzunehmen. Darauf basierend kann dann leicht der Ist-Zustand einzelner Mitarbeiter oder einer Organisation ermittelt werden (Assessment).

In einem weiteren Schritt würde dann ein Ziel-Szenario (Soll) beschrieben werden, anhand dessen gegebenenfalls Ausbildungsmaßnahmen, Rollenbeschreibungen etc. erarbeitet oder geschärft werden.

6.2 Software-Produktlinien

Das aktuellste und möglicherweise auch wichtigste Konzept der Industrialisierung sind Software-Produktlinien (Software Product Lines). Sie spiegeln das industrielle Prinzip der Spezialisierung wider. Es scheint schwierig oder sogar unmöglich zu sein, zu definieren, wie Mechanismen der Wiederverwendung und der Automatisierung in einem mehrdeutigen Kontext implementiert werden können. Systematische Wiederverwendung muss geplant werden und kann nicht zufällig erfolgen. Dem entgegen spannt eine Software Product Line einen klar abgegrenzten Rahmen um eine Familie von fachlich miteinander verwandten Produkten. Diese verfügen über eine Menge gleicher Features und Lösungsansätze einer bestimmten fachlichen Domäne [ClN007, p. xix]. Das Konzept erfordert ferner die Trennung von Produktion und Infrastruktur. Während in ersterem Fall die tatsächlichen Softwareprodukte implementiert werden, stellt die Infrastruktur einer Software Product Line die dafür erforderlichen Komponenten, Architekturen, Frameworks, Werkzeuge, Prozesse und andere wiederverwendbare Artefakte zur Verfügung. Durch die Konzentration auf einen klar abgegrenzten Anwendungsbereich sind diese weitaus mächtiger und leichter wiederzuverwenden als generische Komponenten. Eine derart enge Spezialisierung würde allerding wenig Spielraum für kundenspezifische Anforderungen lassen. Software Product Lines identifizieren daher neben ständig wiederkehrenden Funktionalitäten auch Variationspunkte, die von Kunde zu Kunde unterschiedlich

implementiert werden können. Dieses Konzept ist auch unter der Bezeichnung „Mass Customization“ in bestimmten Industriezweigen bekannt.

Während der eigentlichen Produktentwicklung werden Wissen und wiederverwendbare Artefakte gesammelt und für zukünftige Produkte in die Software Product Line integriert. Die folgende Abbildung verdeutlicht die Zusammenhänge:

Product-Line-Entwickler entwickeln Produkt- und Produktionsartefakte, die wiederum von Product-Entwicklern genutzt werden, um eine konkrete Software zu implementieren. Während der Produktentwicklung neu entwickelte Artefakte fließen zurück in die Produktlinie. Es ist daher sehr wichtig, dass kundenspezifische Anpassungen immer im Hinblick auf Wiederverwendung entwickelt werden. Weitere positive Effekte finden sich in einer höheren Qualität und kürzerer Entwicklungszeit: Da wiederverwendbare Komponenten vielfach getestet und eingesetzt werden, ist die Wahrscheinlichkeit potenzielle Fehler zu finden weitaus höher als bei herkömmlicher Softwareentwicklung [Pohl+05, p.10]. Zudem kann ein einmal entdeckter Fehler behoben werden, bevor er in einem anderen

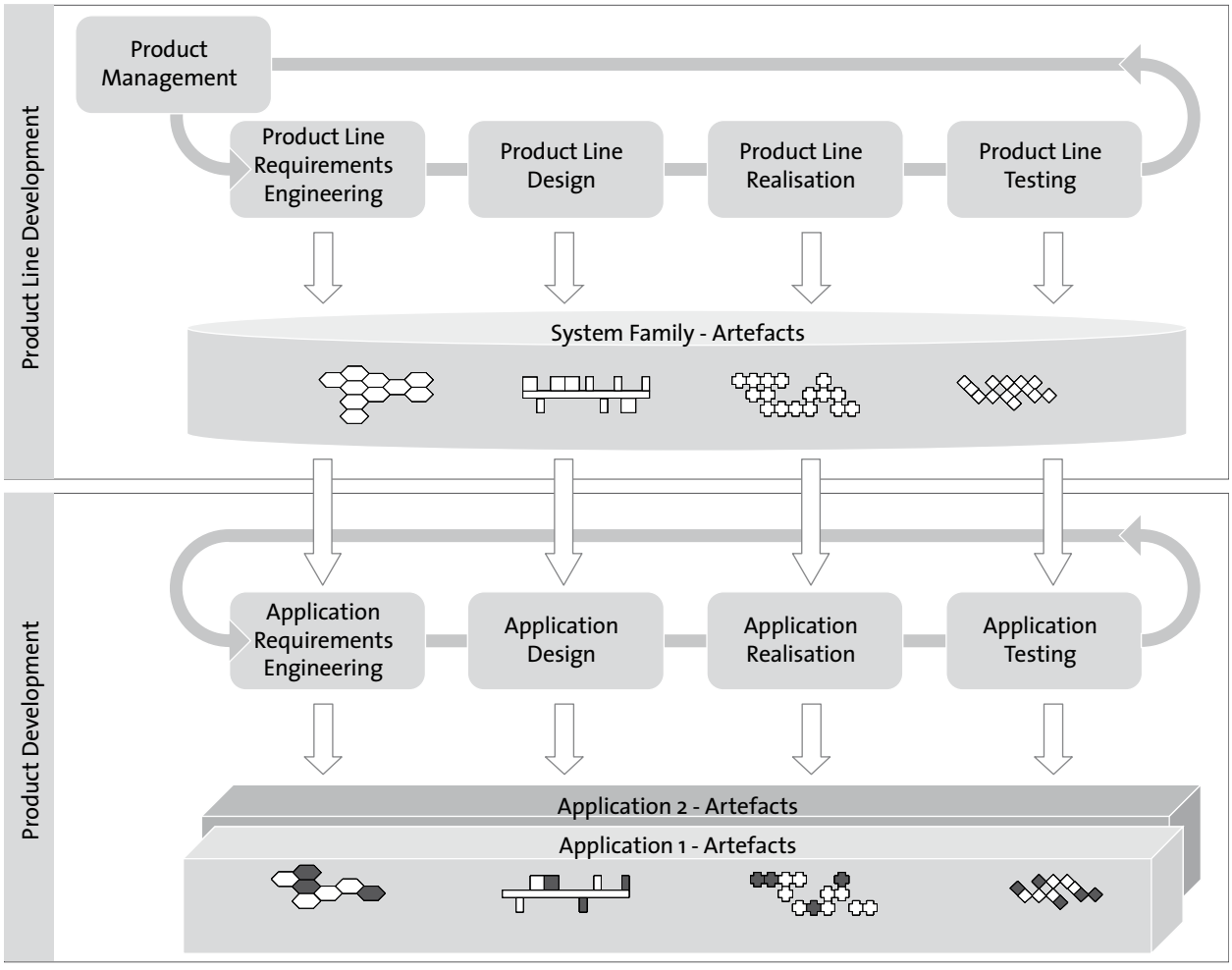


Abbildung 13: Softwareentwicklung innerhalb einer Software-Produktlinie

Produkt auftritt. Durch die vermehrte Wiederverwendung bereits existierender Artefakte reduziert sich die Entwicklungszeit erheblich [Pohl+05, p.10].

Selbstverständlich erfolgt eine Spezialisierung auf ein bestimmtes Segment nicht umsonst. Um einen klar abgegrenzten Anwendungsbereich und eine initiale Menge wiederverwendbarer Artefakte zu entwickeln, sind anfängliche Investitionen unumgänglich. Anders als bei der produzierenden Industrie ist deren Ausgleich aufgrund der leichten Kopierbarkeit von Software nicht durch Skaleneffekte zu erzielen. Software Product Lines müssen sich daher auf sogenannte „Economies of Scope" oder auch inhaltliche Skaleneffekte konzentrieren. Die Produktlinie produziert daher eigenständige, aber sehr ähnliche Produkte, die alle auf einer Menge gemeinsamer Funktionalität beruhen. Gemäß der aktuellen Literatur ist die Entwicklung von ca. drei Systemen erforderlich, um die Anfangsinvestitionen auszugleichen.

6.3 Verteilte Entwicklung und intelligentes Sourcing

Unabhängig von den verschiedenen methodischen und technischen Industrialisierungsansätzen bleibt Softwareentwicklung in Teilen ein kreativer Prozess, der nicht ohne manuelle Arbeitsschritte durch Menschen auskommt (siehe Abschnitt 3.3). Beginnend bei der Aufstellung, Abstimmung und Interpretation von Anforderungen über Architekturentscheidungen bis hin zur Analyse von Testdaten. Überall werden Fähigkeiten benötigt, die uns Computer bisher nicht bieten.

In den verschiedenen Schritten der Entwicklung sind auf der einen Seite unterschiedliche Erfahrungswerte und Bildungsniveaus notwendig. Auf der anderen Seite ist die menschliche Arbeit bei der Softwareentwicklung der größte Kostenfaktor. Vor dem Hintergrund der Globalisierung und dem damit verbundenen Druck zur Ressourcen-Effizienz ist in den vergangenen zehn Jahren das Outsourcing von Softwareentwicklung und Wartung sowie dem Software-Betrieb entstanden. Diesem Ansatz liegt die Übergabe von Aufgaben an Partner zugrunde, welche sich auf deren Lösung spezialisiert haben und dadurch zu geringeren Kosten arbeiten. Offshore-Outsourcing ergänzt diesen Ansatz um die Dimension des globalen Lohngefälles durch Verschiebung von Arbeitsschritten, die sehr zeitintensiv sind und mit geringerem Ausbildungsniveau geleistet werden können, in Länder mit niedrigerem Lohnniveau.

Wenn auch bis heute mitunter kontrovers diskutiert, so ist dieser Trend nicht mehr umkehrbar. Jedoch wird mit der zunehmenden Erfahrung deutlich, dass die richtige Verteilung von Arbeitsschritten auf die verfügbaren Ressourcen in ihrer Komplexität nur schwer zu beherrschen ist. Von der richtigen Ressourcenverteilung (intelligentes Sourcing) hängt der Projekt- und Unternehmenserfolg maßgeblich ab. Abbildung 14 stellt die wesentlichen Prozesse für die Softwareentwicklung vereinfacht dar. Jeder Teilprozess kann bei passenden Rahmenbedingungen durch spezialisierte Partner geleistet werden.

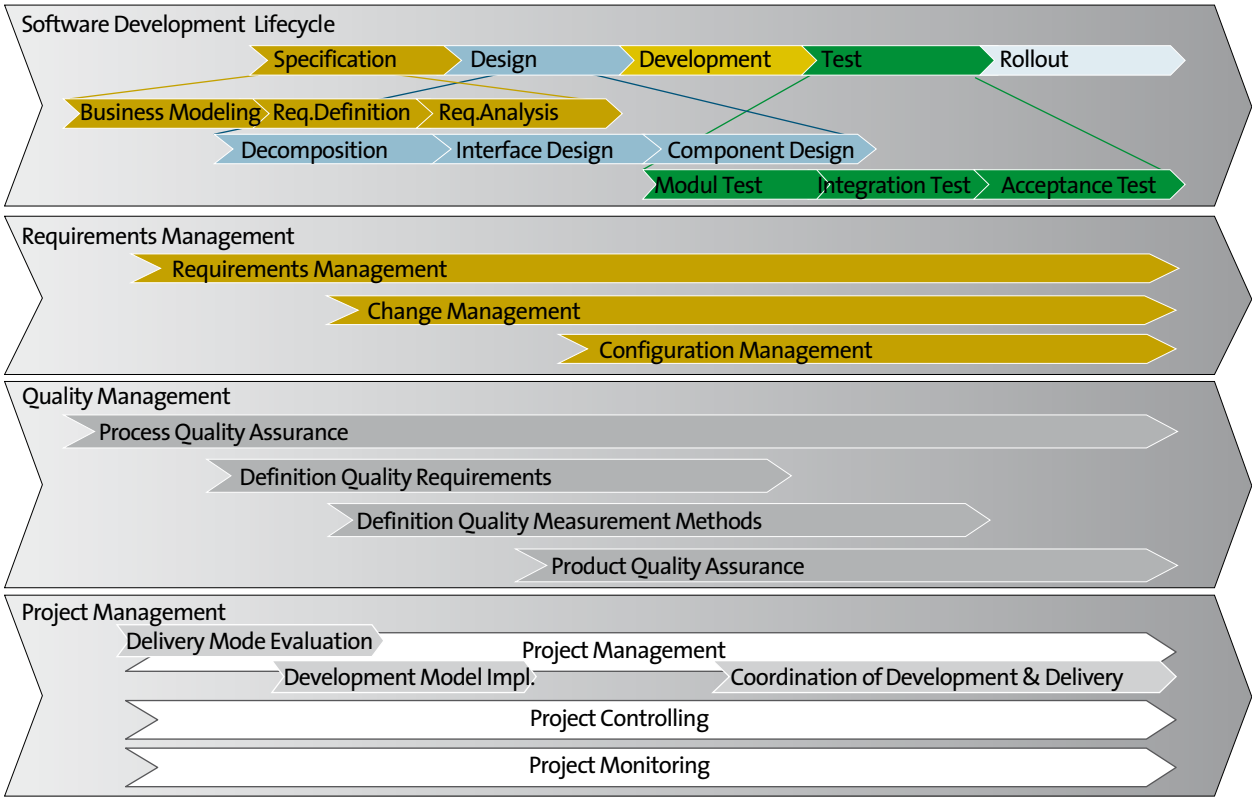


Abbildung 14: Der Softwareentwicklungsprozess und wichtige Unterstützungsprozesse

Unter intelligentem Sourcing ist die ökonomisch optimale Verteilung von Ressourcen auf die notwendigen Arbeitsschritte bei der Softwareerstellung und Wartung unter Berücksichtigung unterschiedlicher Leistungs- und Lohnniveaus zu verstehen. Zusätzlich ist dabei eine Vielzahl von Rahmenbedingungen zu berücksichtigen – angefangen bei unterschiedlichen (Unternehmens-)Kulturen über IP-Schutz bis zu steigenden Kommunikations- und Reisekosten.

Ergebnis der Spezialisierung ist die verteilte Softwareentwicklung. Diese – egal ob nearshore oder offshore – führt zu Herausforderungen und zusätzlichen Risiken in Softwareentwicklungsprojekten. Je nach Projektkonstellation sind die verschiedenen Risikofaktoren unterschiedlich hoch. Bei der räumlich und zeitlich verteilten Softwareentwicklung über Organisations- und Kulturgrenzen hinweg wurden in den vergangenen Jahren vielseitige Varianten der Arbeitsteilung erprobt, verschiedenste Schwachpunkte identifiziert und dafür teilweise Lösungen entworfen. Bevor aber diese Lösungen und Maßnahmen zur Risikominimierung angewendet werden können, müssen die im konkreten Projekt tatsächlich vorhandenen Herausforderungen identifiziert werden.

Dimensionen der Verteilung

Startpunkt der Analyse ist die Verteilungscharakteristik, d. h., wie stark sind die Dimensionen der verteilten Softwareentwicklung ausgeprägt und wie stehen diese in Relation zu bekannten Projektkennzahlen wie Projektlaufzeit, Projektvolumen und Qualitätsanforderungen oder Vertragsstrafen.

Wichtige Dimensionen der Verteilungscharakteristik sind:

- Anzahl der Entwicklungsstandorte und der verteilten Teams
- Anzahl der Beteiligten verschiedener Unternehmen
- ggf. verschiedene Organisationsstrukturen
- kulturelle Unterschiede der beteiligten Teams
- weit entfernte Kunden und Zeitverschiebung
- zentrale oder verteilte Koordinierung
- Informations- und Kommunikationsbedarf
- Schnittstellen und Übergabepunkte in der Wertschöpfungskette

Alle aufgeführten Aspekte sind beim intelligenten Sourcing zu berücksichtigen. Sie beeinflussen das Ressourcen- und Projektmanagement, aber auch den richtigen Zuschnitt des Entwicklungsprozesses. Starre Prozesse und Aufgabenteilungen für sehr unterschiedliche Projektkonstellationen wirken sich allgemein negativ aus. Rein agile Verfahren können bei großen räumlich verteilt arbeitenden Teams wegen der hohen Kommunikation auch nicht angewandt werden. Die Auswahl der geeigneten Übergabepunkte und Entwicklungsartefakte ist ebenfalls betroffen. Für verteilte Projekte ist somit in der Regel ein projektspezifischer Mix aus verschiedenen Vorgehensweisen – abgestimmt auf die Projektstruktur – erforderlich. Im Extremfall ist ein solcher Mix für jedes Projekt erneut zu bestimmen. Die Abbildungen 15 bis 17 zeigen beispielhaft die Verteilung der verschiedenen Teilprozesse und Entwicklungsdisziplinen auf drei Standorte mit unterschiedlichen Lohnniveaus.

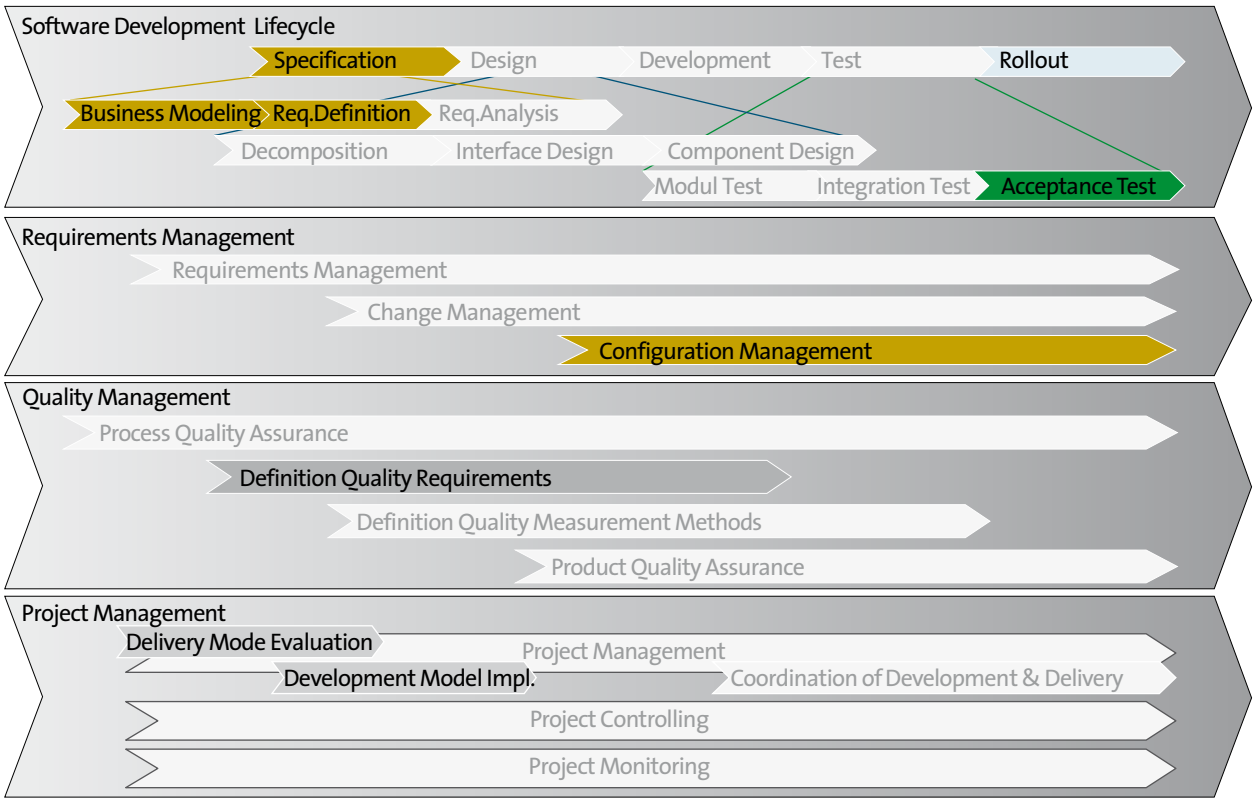


Abbildung 15: Delivery Model – Teilprozesse Customer Site

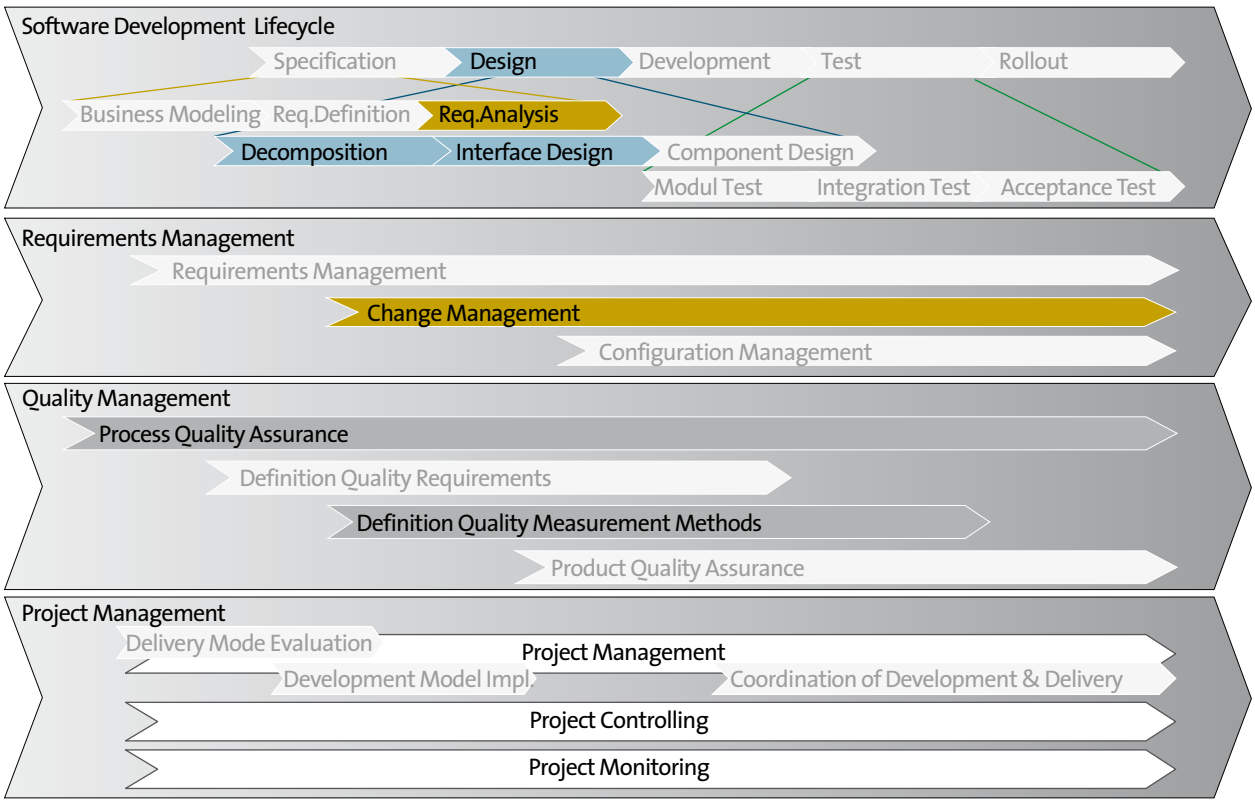


Abbildung 16: Teilprozesse High Cost Country

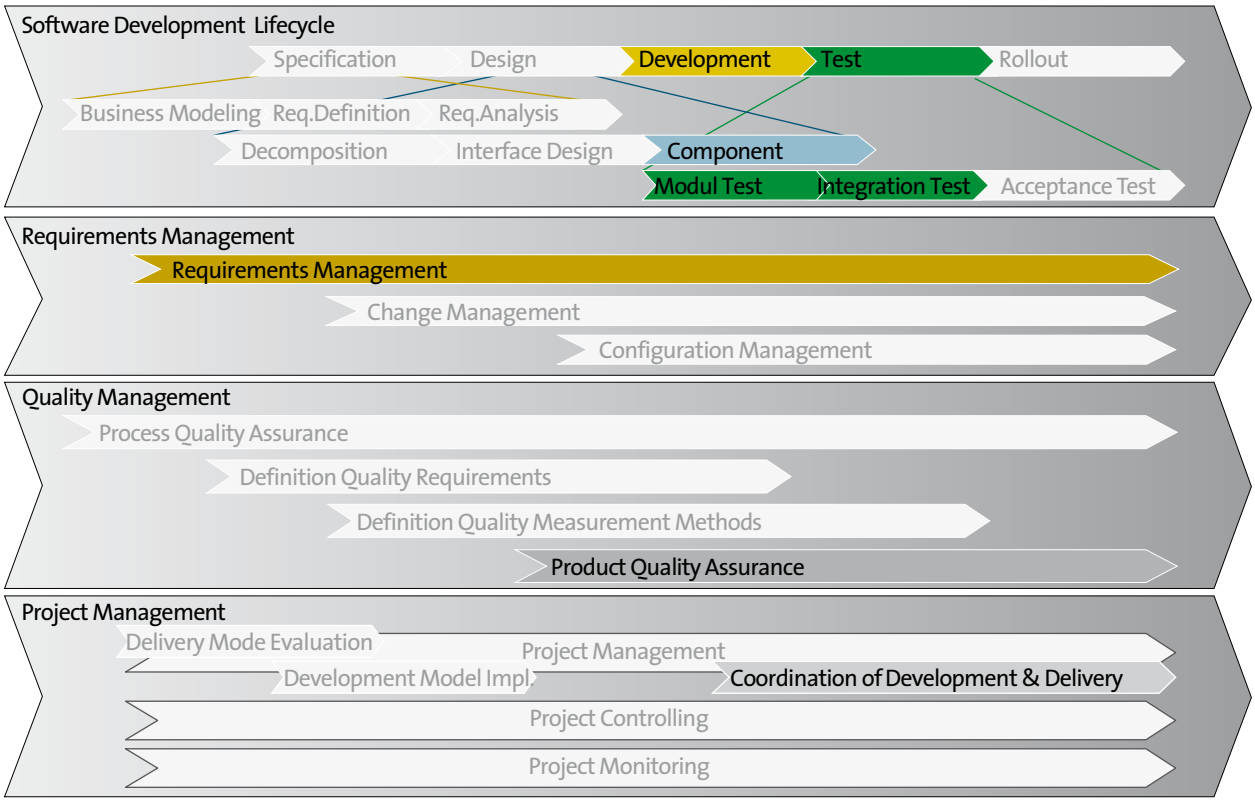


Abbildung 17: Teilprozesse Low Cost Country

Informations- und Kommunikationsbedarf

Eine in der Vergangenheit häufig unterschätzte Frage ist der Informations- und Kommunikationsbedarf und die damit verbundenen Mehrkosten in der verteilten Softwareentwicklung. Sind für die Entwicklung notwendige Informationen nicht in der richtigen Qualität verfügbar, ergeben sich Verzögerungen oder Fehlinterpretationen. Die Folge besteht in falschen Entwicklungen, die ein großes Risiko darstellen, da Korrekturmaßnahmen Kosten verursachen können, die den wirtschaftlichen Vorteil infrage stellen können. Die Wahl der geeigneten Entwicklungsartefakte und Kommunikationsmittel ist damit ein wesentlicher Bestandteil von intelligentem Sourcing. Werkzeuge wie Kollaborationsplattformen, Audio- und Video-Telefonie und gemeinsame Repositories sind bereits heute verfügbar, können den Informationsfluss unterstützen, entbinden aber nicht von der Definition von Entwicklungsartefakten und Übergabenpunkten. Weitere Optionen der verteilten Zusammenarbeit in der Softwareentwicklung, z. B. gemeinsames verteiltes Modellieren, sind heute noch Gegenstand der Forschung.

6.4 Ganzheitlicher Ansatz (Software Factories)

Der maximale Grad der Industrialisierung von IT-Projekten entlang der Dimension „Spezialisierung“ besteht in der Nutzung entsprechender Software Factories, im Idealfall angeboten durch einen Dienstleister. Die Industrialisierung zeigt in Software Factories also bereits Formen der Deindustrialisierung (s. o.).

Unter einer Software Factory wird im Folgenden verstanden:
„A software product line that provides a production facility for the product family by configuring extensible tools using a software template based on a software schema.“ [Gree10]

Software Factories sind für solche Unternehmen im Kontext der Industrialisierung besonders wichtig, die Softwareentwicklung nicht als Kernkompetenz anbieten. So wie der Verbraucher die Erzeugung vieler Güter kontinuierlich durch Spezialisten durchführen lässt und rein ergebnisorientiert vorgeht, so können auch Software Factories diese Sicht ermöglichen. Die Software Factories können im Gegenzug aufgrund der Vielzahl ähnlicher Anfragen Software-Produktlinien aufbauen.

Eine sehr wichtige Vorbedingung, um dies dann tatsächlich durchführen zu können, ist allerdings ein möglichst ganzheitlicher Ansatz bezüglich der geforderten Ergebnisse. Es genügt heute für die Verwendung einer Software Factory nicht, eine formulierte Funktionalität zu übergeben und als Ergebnis das fertige Produkt ausgeliefert zu bekommen. Vielmehr müssen die Anforderungen an das zu erstellende Produkt ganzheitlich formuliert werden, damit das gelieferte Produkt als qualitativ hochwertig bewertet werden kann. Wird als Definition von Qualität die ISO8402 angenommen („Quality: the characteristics of a product or service that bear on its ability to satisfy stated or implied needs“) so sollte die Ganzheitlichkeit entlang der beiden folgenden Dimensionen erfolgen:

- Produktkatalog: Was sind die konkreten Ergebnistypen, die die Software Factory liefern soll? Hier genügt nicht die reine Lieferung eines fertigen ausführbaren Software-Moduls. Weitere wichtige Ergebnistypen sind z. B. Dokumentation, Konfiguration, Source Code, Modelle, Testfälle usw. In der Microsoft Software Factory werden diese Artefakte im sogenannten Software-Factory-Schema festgehalten (vgl. [Gree10]).
- Anforderungskatalog: Was sind die jeweiligen Anforderungen, die an ein spezifisches Produkt gestellt werden? Eine typische Anforderung an das fertige ausführbare Software-Modul ist beispielsweise eine korrekte Funktionalität. Weitere Anforderungen können aber z. B. Effizienz, Wartbarkeit oder auch Portierbarkeit sein. Auch an Dokumentationen und Testfälle können spezifische Anforderungen gestellt werden. Nur wenn diese im Vorfeld spezifiziert werden, kann die Software Factory auch entsprechend dieser Vorgaben arbeiten.

Im Folgenden werden einigen Beispiele für erfolgreiche Einsätze von Software Factories dargestellt.

6.4.1 Software-Wartung/Test Factories

Je länger ein Softwaresystem eingesetzt worden ist, desto höher ist die Wahrscheinlichkeit, dass es geändert werden muss. Diese Änderungen können motiviert sein durch:

- Anpassungen und Änderungen der Anforderungen (ganzheitlich bgzl. ISO 9126, z. B. geänderte Anforderungen bzgl. Performance)
- Anpassungen und Änderungen der Infrastruktur (Betriebssystem, Umsysteme, Hardware)
- notwendige Korrekturen bei aufgedeckten Fehlern

Diese Aktivitäten können für Softwaresysteme durchaus die dominanten Kostentreiber sein und 75 Prozent der Total Cost of Ownership ausmachen.

Die Industrialisierung und insbesondere Software Factories zeigen hier ihre großen Vorteile. Wesentliche Kostentreiber der Wartung sind i. d. R.:

- das Verstehen der bisherigen Umsetzung
- das Lokalisieren notwendiger Änderungsmaßnahmen darin
- die Durchführung der Maßnahmen selbst sowie
- der anschließende Test

Für all diese Aktivitäten werden unterschiedliche Software-Artefakte verwendet (z. B. Architekturen für das Lokalisieren), für die für einen effektiven Einsatz grundsätzlich eine maximale Kongruenz zum tatsächlichen System gefordert wird. Demzufolge besteht eine Wartungsaktivität nicht nur aus der Anpassung des Softwareproduktes selbst, sondern ebenfalls aus der Anpassung und Konsistenzsicherstellung aller beteiligten Zwischenprodukte. Die Ergebnisse einer Industrialisierung helfen hier enorm, da es wohldefinierte Pakete (Modularisierung) mit spezifischen Aufgaben gibt.

Insbesondere Software Factories spielen hier ihren Mehrwert aus, da ein Großteil der Zwischendokumente (semi-) automatisch erzeugt und aktualisiert wird und die Wartung so deutlich effizienter durchgeführt werden kann. Produktlinien ermöglichen daher ein allumfassendes Application Lifecycle Management (ALM), bei dem alle relevanten Produkte fortwährend up to date gehalten werden und damit Wartung langfristig wirtschaftlich interessant gestaltet werden kann.

Insbesondere für den Softwaretest kommen immer häufiger sogenannte Test Factories zum Einsatz: Auch hier helfen Industrialisierungskonzepte, in dem die dort geforderte Modularisierung z. B. die deutliche Trennung folgender Aspekte ermöglicht:

- Testfälle (z. B. die zu testenden Business-Prozesse)
- Testdaten (z. B. die konkreten Eingabedaten für eine Berechnungsmaske)
- GUI-Objekte (z. B. die konkreten Eingabefelder einer Maske)
- und Automatisierungswerkzeuge

Ein Re-Test nach einer Wartungsänderung kann so zu einem maximal hohen Maß wiederverwendet werden, indem nur diejenigen Objekte der Testumgebung angepasst werden (z. B. neue GUI-Elemente), die durch die Wartungsaktivität betroffen sind. Diese neuen Industrialisierungsaspekte des Softwaretestens führen aktuell zu neuen Märkten wie Cloud Testing (vgl. IBM testing in the cloud) oder Managed Testing Services (vgl. SQS). Beide Richtungen ermöglichen die effiziente und effektive Durchführung des (Re-)testens eines Systems für Auftraggeber und Auftragnehmer. Die Auftragnehmer haben sich dabei stark auf den spezifischen Service-Bereich fokussiert und hosten eine Vielzahl von notwendigen Test-Umgebungen, Automatisierungswerkzeugen, Testdatenbanken, Testdatenanonymisierern und Testauswertern, die insgesamt als Testware bezeichnet werden. Auch hier zeigt sich, dass die Spezialisierungen nicht autark, sondern im Zusammenspiel mit den anderen Industrialisierungsdimensionen gesehen werden muss. Managed Testing Services stellen z. B. eine sehr reife Form der Spezialisierung dar, bedienen sich aber intern natürlich der Automatisierung, der Wiederverwendung und der Standardisierung.

6.5 Offshore-Outsourcing – einen Schritt zu früh?

Mit Blick auf verschiedene Branchen lässt sich feststellen, dass diese zunächst industrielle Produktionsmethoden, basierend auf Spezialisierung, Standardisierung, Wiederverwendung und Automatisierung, eingeführt haben. Erst nach deren erfolgreicher Umsetzung, und damit der klaren Abgrenzbarkeit bestimmter Herstellungsschritte, wurde damit begonnen, einfache und arbeitsintensive Produktionsprozesse in Niedriglohnländer zu verlagern. Der Vergleich mit der Softwarebranche zeigt jedoch ein genau umgekehrtes Vorgehen. Hier wurde zuerst versucht, die eigentliche Implementierung von Anwendungen in Niedriglohnländer wie Indien zu verlagern, während Design und Projektmanagement im jeweiligen Inland verblieben. Erst jetzt werden auch Ansätze industrieller Produktionsmethoden eingeführt, wie beispielsweise Software Product Lines, Component-based Development oder Model-driven Engineering. Die teils mäßigen oder gar schlechten Erfahrungen, die manche Unternehmen mit Offshore-Outsourcing von Softwareentwicklung gemacht haben, lassen auf eine unzureichende Eignung der jeweiligen Projekte schließen. Dies wird insbesondere vor dem Hintergrund der in der Literatur identifizierten kritischen Erfolgsfaktoren für Offshore-Projekte ersichtlich. Am wichtigsten erscheinen dabei jene Aspekte, die den industriellen Grundprinzipien Spezialisierung und Standardisierung zuzuordnen sind:

- Der Begriff Spezialisierung im gegebenen Kontext beschreibt die Konzentration einer wirtschaftlichen Entität (Arbeiter, Unternehmen, Gesellschaft etc.) auf ein abgegrenztes Gebiet innerhalb ihres Wirtschaftsraums. Ein solches Gebiet kann z. B. eine bestimmte Industrie, eine Produktlinie oder auch ein bestimmtes Know-how sein. Hierdurch wird der Produktionsprozess in kleine, weniger komplexe Funktionen unterteilt, die von speziell ausgebildeten Mitarbeitern oder Maschinen durchgeführt werden können. Diese Form der Arbeitsteilung erlaubt die effizientere und qualitativ hochwertigere Umsetzung dieser Funktionen, erfordert jedoch eine exakte Abgrenzung des jeweiligen Aufgabenbereichs. Ist diese gegeben und sind auch die Schnittstellen zwischen ihnen klar definiert, können bestimmte Bereiche an externe Dienstleister ausgelagert werden. Das Fehlen einer solch klaren Abgrenzung und unsaubere Schnittstellen werden als Hauptursachen für scheiternde Offshore-Projekte genannt [Bitko5].
- Standardisierung spielt eine weitere wichtige Rolle im Industrialisierungsprozess. Sie beschreibt die Vereinheitlichung bestimmter Attribute von Produkt- oder Produktionsartefakten. Ziel hierbei ist eine gemeinsame Sichtweise, um Teilprodukte untereinander auszutauschen oder zu integrieren. Kritisch bei der Standardisierung ist eine kontextfreie Beschreibung der Anforderungen. Insbesondere mit Hinblick auf kulturelle Unterschiede ist manche in unseren Augen selbstverständliche Antwort in einem anderen Kulturkreis bei Weitem nicht selbstverständlich. Diese kontextfreie Spezifikation stellt einen weiteren kritischen Erfolgsfaktor für Offshore-Projekte dar. Dazu kommen Unterschiede in den Entwicklungsprozessen zwischen beiden Partnern. Oft arbeitet beispielsweise ein indisches Unternehmen nach CMMI Level 5, währen im Inland lediglich Level 2 oder 3 erreicht wurde.

Es scheint somit fraglich, ob Offshore-Outsourcing ohne eine zumindest ansatzweise industrielle Softwareentwicklung erfolgreich sein kann. Umgekehrt hingegen werden die wesentlichsten kritischen Erfolgsfaktoren abgedeckt und weitere Vorteile erschlossen.

Software Product Lines sorgen für eine klare Abgrenzung der verschiedenen Entwicklungsprozesse und klar definierte Schnittstellen und Zuständigkeiten. Es erfolgt eine Trennung zwischen ingenieurmäßig entwickelnden und produzierenden Tätigkeiten. Dabei definiert der begrenzte Anwendungsbereich einer Produktlinie auch den für die Entwickler erforderlichen Kontext. Darüber hinaus lässt sich wertvolles Know-how über Design und Architektur einer Produktfamilie von der eigentlichen Entwicklung des Produkts trennen. Die Implementierung der Produktlinie mit ihren Kernbestandteilen wird mit gut ausgebildeten Fachkräften im eigenen Unternehmen und am lokalen Standort durchgeführt, während die spätere Entwicklungsarbeit unter Zuhilfenahme der Produktlinie und ihrer Kernbestandteile in Niedriglohnländern

durchgeführt werden kann. Dieses Vorgehen kann in vielen Industrien festgestellt werden. Die Entwicklung und Konstruktion komplexer Produkte erfolgt im Inland, während die eigentliche Produktion in Niedriglohnländer ausgelagert wird. Das oft beobachtete Kopieren geistigen Eigentums lässt sich im Bereich der Softwareentwicklung durch geschicktes Rechtemanagement oder die Nutzung von Grey- oder Blackbox-Prinzipien verhindern oder zumindest erschweren.

Component-based Development stellt die zweite sinnvoll erscheinende Technologie im Rahmen von Offshore-Outsourcing dar. Mithilfe von Komponentenframeworks und Systemarchitekturen kann eine Anwendung sinnvoll modularisiert werden. Damit können klar abgegrenzte Entwicklungsaufgaben an externe Anbieter vergeben werden. Die Nutzung standardisierter Komponentenframeworks bietet jedoch noch einen weiteren Vorteil. Die engen Grenzen, die einer Komponente durch das zugrunde liegende Framework bzw. die Architektur gesetzt werden, reduzieren den späteren Integrations- und Korrekturaufwand erheblich.

Eine weitere Voraussetzung für erfolgreiche Offshore-Projekte ist eine kontextfreie Beschreibung der Anforderungen. Jedweder Interpretationsspielraum kann zu Abweichungen vom eigentlich benötigten Funktionsumfang führen. Die Erstellung einer solchen Beschreibung ist jedoch bei Weitem nicht trivial: Hier kommen Ansätze aus Software Product Lines, Component-based Development und Model-driven Engineering zusammen. Produktlinien sorgen für einen klar abgegrenzten Anwendungsbereich und schränken die möglichen Kontexte bereits stark ein. Systemarchitekturen und Komponentenframeworks sorgen für eine weitere Abgrenzung über die Anforderungen an eine Komponente. Letztlich erlaubt jedoch Model-driven Engineering unter Zuhilfenahme von domänenspezifischen Sprachen (DSLs) eine weitaus genauere Beschreibung der Anforderungen. DSLs bestehen aus eigens definierten sprachlichen Elementen, die für die Beschreibung eines bestimmten Anwendungsbereiches erforderlich sind, und spezifizieren diesen dann weitaus einfacher als es beispielsweise UML oder gar natürliche Sprachen könnten.

Betrachtet man die eingangs erwähnten Erfolgsfaktoren für Offshore-Projekte, so lässt sich erkennen, dass diese in den Konzepten industrieller Softwareentwicklung weitgehend abgedeckt werden. Ähnlich verhält es sich in anderen Branchen, die i. d. R. zunächst industrialisiert und erst dann Produktionsschritte an externe Anbieter ausgelagert haben. Es erscheint daher wenig sinnvoll, Offshore-Outsourcing vor der Industrialisierung der Softwareentwicklung einzuführen.

6.6 Fazit Spezialisierung

Spezialisierung als Industrialisierungsansatz ist richtig eingesetzt ein mächtiges Werkzeug für die Softwareentwicklung. Dabei reicht die Palette der konkreten Ausprägungen von technologiegetriebenen Entwicklungslinien über produktzentrierte Software-Produktlinien bis zum kostenfokussierten Verteilen der verschiedenen Entwicklungsaufgaben auf unterschiedliche Teams. Den maximalen Grad der Industrialisierung von IT-Entwicklungsprojekten im Sinne der Spezialisierung stellen Software Factories dar, bei denen im Idealfall viele Schritte der Wertschöpfung durch Dienstleister erbracht werden.

Im Rahmen der Globalisierung der IT-Industrie wurde in den vergangenen zehn Jahren vor dem Hintergrund des Fachkräftemangels in der ersten Welt und des erheblichen weltweiten Einkommensgefälles Offshore-Outsourcing populär. Zentrales Element ist hier die Wahl einer betriebswirtschaftlich motivierten Sourcing-Strategie. Im Gegensatz zum Betrieb von Software – dem der Entwicklung folgende Abschnitt des Software Lifecycle – ergeben sich bei der Softwareerstellung aber zusätzliche Herausforderungen. Die im Offshore-Outsourcing für Entwicklungsprojekte kalkulierten Einsparpotenziale können beispielsweise durch unterschätzte Kommunikationsaufwände oder nicht hinreichend spezifizierte Anforderungen schnell zunichte gemacht werden. Die Wahl diese Delivery Models sollte daher nur unter Berücksichtigung aller anderen Industrialisierungsansätze getroffen werden.

7 Continuous Improvement

7.1 Maturity Models – Brückenschlag zwischen Prozess-/Produktstandard und Organisation

Maturity Models wie das Capability Maturity Model Integrated (CMMI) der SEI basieren auf der Einführung von Prozessstandards und entsprechenden Methoden für deren Umsetzung im Unternehmen. Aus Letzterem ergeben sich unterschiedliche Reifegrade (Capability oder Maturity Levels), die ein Unternehmen haben kann. Beispiel: Ein Unternehmen, das in jedem Entwicklungsprojekt ein professionelles, aber individuelles Vorgehen anwendet, befindet sich im Reifegrad 1. Ein Unternehmen, das einheitliche Prozesse für das Management und die Umsetzung von Entwicklungsprozessen hat, diese kontinuierlich verbessert und dabei gesammeltes Wissen aus dem Unternehmen wiederverwendet, befindet sich im Reifegrad 3. Ab hier finden wir zahlreiche Elemente der Industrialisierung auf Prozess- und Organisationsebene.

Auf der Prozessebene fordert CMMI beispielsweise:

- professionelle und standardisierte Softwareentwicklungsmethoden von der Anforderungsanalyse bis zur Systemeinführung
- konsistente Planung, Nachverfolgung (Controlling) und Management von Projekten
- durchgängiges Anforderungs- und Konfigurationsmanagement (Traceability)
- durchgängige Qualitätssicherung, im Speziellen Reviews und Test

7.2 Verankerung im Unternehmen

Maturity Models wie CMMI unterstützen ein Unternehmen dabei, die Prozessstandards organisatorisch zu verankern. Erst gelebte und kontinuierlich verbesserte Standards ermöglichen es, von einer Effizienzsteigerung der Industrialisierung zu profitieren und sich gleichzeitig immer wieder an veränderte Anforderungen und Rahmenbedingungen anzupassen (Markt und Technologie).

Dazu sind auf unterschiedlichen Ebenen verschiedene Maßnahmen erforderlich.

7.2.1 Methoden und Werkzeuge

In einem Unternehmen eines höheren Reifegrades werden standardisierte Methoden und Werkzeuge in einer einheitlichen Form eingesetzt. Es werden Vorgehensmodelle und die dafür passenden Werkzeuge definiert. Das schließt nicht aus, dass abhängig von Anwendungsbereich, Kundensegment und Technologie unterschiedliche Standards zum Einsatz kommen – im Gegenteil: Das ist ein Zeichen für einen hohen Industrialisierungsgrad. Für jeden Einsatzbereich sind die Standards jedoch vorgegeben und auf die Anforderungen des Unternehmens spezialisiert.

7.2.2 Personelle Maßnahmen

Die personellen Maßnahmen beziehen sich vor allem auf die Schulung der Mitarbeiter hinsichtlich der Standardmethoden und -werkzeuge, die zielgerichtete Auswahl von Mitarbeitern (und anderen „Ressourcen“) nach Bedarf und Qualifikation und das Einbeziehen der Mitarbeiter in einen kontinuierlichen Verbesserungsprozess.

7.2.3 Organisatorisch

Organisatorische Maßnahmen schließen ein, dass ein Unternehmen zunächst definiert, wie es arbeitet (Unternehmensstandards), wie es diese Standards weiterentwickelt und verbessert und wie es deren Umsetzung und Einhaltung forciert. Dazu zählen:

- Unternehmensleitlinien/-policies (z. B. zur Verankerung der anzuwendenden Standards)
- Projekt-, Prozess- und Management-Überprüfungen (Reviews, Quality-Gates, Audits)

- Einführung und Umsetzung eines Messsystems (Metriken)
- ein kontinuierlicher Verbesserungsprozess (inklusive eines funktionierenden Wissensmanagements)

Der kontinuierliche Verbesserungsprozess muss sowohl Erfahrungen und Verbesserungsvorschläge aus der unmittelbaren Anwendung von Methoden und Werkzeugen berücksichtigen als auch Ergebnisse aus Qualitätsüberprüfungen und Messungen.

7.3 Metriken und Messbarkeit

Gemessen werden können Prozessqualität und Produktqualität. Dazu werden spezifische Metriken oder Key Performance Indicators (KPIs) definiert und eingeführt.

Das Messen von Prozessqualität in der Softwareentwicklung erfolgt entlang des Entwicklungsprozesses (Lifecycle). Beispiele sind die Stabilität von Anforderungen, die Verteilung von Fehlern über den Lifecycle, die Durchlässigkeit von Phasen in Bezug auf Fehler sowie klassische Projektmanagement-Metriken wie Termin- und Budgettreue.

Das Messen der Produktqualität erfolgt an den erstellten Ergebnissen zu bestimmten Zeitpunkten des Entwicklungsprozesses. Beispiele dafür sind das Messen der Fehlerdichte in Dokumenten (z. B. nach einem Review der Fachspezifikation), die Abweichung von Codierstandards oder die Anzahl von Fehlern nach dem Integrationstest. Produktmetriken erfordern ein Maß zur Normierung. Beispiele sind Aufwand (aufgrund der fehlenden Vergleichbarkeit eher ungeeignet), Lines of Code, Function Points o. ä.

Generell ist zu beachten, dass Ergebnisse von Messungen mit verschiedenen Metriken gegenüber qualitativen Überprüfungen aufgrund der definierten Messlogik zwar gut vergleichbar sind, jedoch in der Einführung sehr aufwändig sind und deshalb oftmals wenig Akzeptanz im Unternehmen haben. Schlüsselerfolgsfaktor ist hier, ein angemessenes Kosten-/Nutzenverhältnis anzustreben und Messungen möglichst einfach, z. B. werkzeugunterstützt, zu gestalten.

7.4 Fazit Continuous Improvement

Kontinuierliche Verbesserung der Softwareentwicklung bildet die logische Klammer um alle beschriebenen Industrialisierungsansätze. Speziell für die IT-Industrie erarbeitete Maturity-Modelle wie CMMI bilden eine gute Grundlage, um einen unternehmensspezifischen Verbesserungsprozess zu entwerfen, auszurollen und zu leben. Der Prozess berücksichtigt sowohl Erfahrungen und Verbesserungsvorschläge aus der unmittelbaren Anwendung von Methoden und Werkzeugen als auch Ergebnisse aus Qualitätsüberprüfungen und Messungen. Messbarkeit und Metriken sind daher ein wesentlicher Bestandteil des Continuous Improvement. Erfolgskritisch ist auch die unternehmensweite Verankerung von Verbesserungsprozessen im Sinn der Unternehmenskultur.

8 Fazit

IT-Projekte werden heute nicht mehr primär als kreative, nicht steuerbare Prozesse angesehen, deren Hauptherausforderung es ist, die Grenzen technischer Machbarkeit sukzessive zu überspringen. Vielmehr folgt dem Bewusstsein, dass technisch grundsätzlich fast alles machbar ist, die Anforderung das Machbare effizienter und effektiver zu erreichen.

Ein wesentlicher Ideengeber hierfür wird in diesem Leitfaden beschrieben: Industrialisierung bzw. IT-Industrialisierung. Hierbei wird weniger das hohe Lied der Industrialisierung fortgesungen – so wird Industrialisierung im aktuellen Trendradar von Arthur D. Little immerhin als „Mega-Trend" bezeichnet, der Einfluss auf die gesamte Anwendungsentwicklung und den kompletten IT-Betrieb hat, vgl. [DöJu09, pp.6-13] – als vielmehr versucht, den Begriff zu analysieren, Parallelen zu anderen Disziplinen herzustellen und jeweils darzustellen, wo der Stand der Technik für die IT heute steht.

In diesem Leitfaden wurden als Ergebnis die folgenden fünf Industrialisierungsdimensionen vorgestellt:

- Standardisierung
- Automatisierung
- Wiederverwendung
- Spezialisierung
- kontinuierliche Verbesserung

Die IT kann bereits heute in allen fünf Dimensionen praxistaugliche Ergebnisse vorweisen, was pro Dimension durch viele aktuelle, direkt einsatzfertige Beispiele untermauert wird.

Die Analyse anderer Domänen zeigt allerdings auch, dass Industrialisierung mehr ist als das Zusammenstecken von Industrialisierungsfragmenten. Vielmehr zeigt dieser Leitfaden immer wieder notwendige Vorbedingungen und Nachbedingungen (Constraints) auf, deren Nichtbeachtung durchaus zu negativen Effekten in der Softwareentwicklung führen kann. Eine erfolgreiche Industrialisierung besteht demnach nicht in der Optimierung einzelner Industrialisierungsdimensionen, sondern in der ganzheitlichen, gleichwertigen Orchestrierung der unterschiedlichen Dimensionen hin zu einer Gesamtindustrialisierung.

Dieser Leitfaden bleibt bezüglich konkreter Anwendungsempfehlungen auf einem hohen Abstraktionsniveau. Der Leitfaden fordert, die unterschiedlichen Industrialisierungsdimensionen gleichzeitig zu bedienen, bleibt aber u. a. folgende Antworten schuldig:

- Wie groß darf der Fortschritt in einem Bereich sein, bevor alle anderen nachgezogen werden müssen?
- Wie kann innerhalb einer Industrialisierungsdimension möglichst effizient und effektiv die Industrialisierung vorangebracht werden und wie kann dies ggf. gemessen werden?
- Wo liegen die Grenzen der Industrialisierung – sowohl pro Dimension als auch insgesamt?
- Was sind andere „Ideengeber" (s. o.), um das Machbare effizienter und effektiver zu erreichen? Als Beispiel sei hier auf agile Vorgehensweisen verwiesen.
- Wie kohärent sind diese unterschiedlichen Ideengeber? So bleibt z. B. offen, ob die Spezialisierung, typischerweise auch abgebildet auf eine strikte Aufgabenteilung, von agilen Vorgehen explizit angegriffen wird (indem z. B. möglichst flache Organisationen gefordert werden, in denen möglichst jeder alles kann) oder sie diese nur anders darstellen (indem z. B. die agile Vorgehensweise selbst die Spezialisierung darstellt).

Der vorliegende Leitfaden möchte mit diesem ersten Wurf eine Diskussionsbasis inklusive Nomenklatur bereitstellen, die Ausgangslage weiterer Diskussionen zur Beantwortung eben dieser und weiterer Fragen sein kann. Es soll nicht verschwiegen werden, dass bereits dieser Leitfaden sicher kontroverse Diskussionen auslösen wird und soll (vgl. z. B. [DiLe10]). Hierzu wird an dieser Stelle aber explizit eingeladen: Wir haben hierfür als Austauschmedium ein eigenes Diskussionsforum aufgesetzt, in dem die Autoren dieses Leitfadens offen für Anregungen, Kommentare oder Kritiken sind. Ziel ist es, auf Basis

des dann erarbeiteten Ergebnisses die nächsten Schritte gemeinsam zu gehen und den Leitfaden mit konkreteren Best Practices zu untermauern.

9 Quellenverzeichnis

[Gabl10] Gabler Verlag (Herausgeber), Gabler Wirtschaftslexikon, Stichwort: Industrialisierung, online im Internet: http://wirtschaftslexikon.gabler.de/Archiv/55240/industrialisierung-v6.html, Stand: 19.07.2010.

[FrSt07] dPunkt-Verlag, Praxis der Wirtschaftsinformatik: IT-Industrialisierung, August 2007, Heft 256, Herausgeber: Hans-Peter Fröschle, Susanne Strahringer.

[Bren07] vgl. Computerwoche 2007: Walter Brenner, Direktor des Instituts für Wirtschaftsinformatik der Universität St. Gallen.

[Grab98] Grabher, Gernot: De-Industrialisierung oder Neo-Industrialisierung? Innovationsprozesse und Innovationspolitik in traditionellen Industrieregionen, 1998.

[IEEE729] Definition von Software-Engineering des IEEE (Institute of Electrical and Electronics Engineers).

[Broy+06] Broy, Jarke, Nagl, Rombach. Manifest zur strategischen Bedeutung des Software Engineering in Deutschland, 2006.

[FHKS09] Friedrich, Hammerschall, Kuhrmann, Sihling. Das V-Modell XT. Springer, 2. Auflage, 2009.

[KTF10] Kuhrmann, Ternité, Friedrich. Das V-Modell XT anpassen. Springer, (erscheint) 2010.

[KKD09] Marco Kuhrmann, Georg Kalus, Gerhard Chroust. Tool-Support for Software Development Process, Herausgeberin: Maria Manuela Cruz-Cunha, ch. 11, pp. 213-231, Business Science Reference, number ISBN: 978-1-60556-856-7, IGI Global, 2009.

[Szyp99] Szyperski, 1999, S. 27.

[Greu03] Greulich, 2003, S. 278–280.

[Andr04] Andresen, Andreas: Komponentenbasierte Softwareentwicklung mit MDA, UML 2 und XML. 2., neu bearb. Aufl. München: Hanser.

[Greu03] Greulich, Walter (Hg.) (2003): Der Brockhaus Computer und Informationstechnologie. Hardware, Software, Multimedia, Internet, Telekommunikation. Mannheim: Brockhaus.

[Mcil69] McIlroy, Douglas (1969): Mass Produced Software Components. In: Naur, Peter; Randell, Brian (Hg.): Software Engineering. Report on a conference sponsored by the NATO SCIENCE COMMITTEE. Brussels, S. 138–156.

[Szyp99] Szyperski, Clemens: Component software. Beyond object-oriented programming.1999. Harlow: Addison-Wesley.

[Schm06] Schmidt, 2006.

[GrSh04] Greenfield and Short, 2004

[Belt+07] Beltran et al., 2007

[ClNo07] Clements, P. and Northrop, L., 2007, p. xix

[Pohl+05] Pohl, K., Böckle, G., et al., 2005, p. 10

[Bitk05] Leitfaden Offshoring 2005; BITKOM (Hg.), Berlin, 2005.

10 Danksagung

Besonderer Dank gilt dem BITKOM-Arbeitskreis Software Development Processes and Tools, insbesondere den Autoren des Leitfadens:

- Werner Achtert, TÜV Informationstechnik GmbH
- Torsten Becker, BESTgroup Consulting & Software GmbH
- Hubert Biskup, IBM Deutschland GmbH
- Dirk Hellebrand, FOURTH PROJECT GmbH
- Jürgen Herczeg, T-Systems International GmbH
- Stephan Krause, CSC Deutschland Solutions GmbH
- Marco Kuhrmann, Technische Universität München
- Frank Maar, Microsoft Deutschland GmbH
- Frank Marschall, T-Systems International GmbH
- Matthias Minich, T-Systems International GmbH
- Frank Simon, SQS Software Quality Systems AG
- Stephan Ziegler, BITKOM e.V.

Der Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. vertritt mehr als 1.300 Unternehmen, davon 950 Direktmitglieder mit etwa 135 Milliarden Euro Umsatz und 700.000 Beschäftigten. Hierzu zählen Anbieter von Software, IT-Services und Telekommunikationsdiensten, Hersteller von Hardware und Consumer Electronics sowie Unternehmen der digitalen Medien. Der BITKOM setzt sich insbesondere für bessere ordnungspolitische Rahmenbedingungen, eine Modernisierung des Bildungssystems und eine innovationsorientierte Wirtschaftspolitik ein.

BITKOM

Bundesverband Informationswirtschaft,
Telekommunikation und neue Medien e.V.

Albrechtstraße 10 A
10117 Berlin-Mitte
Tel.: 030.27576-0
Fax: 030.27576-400
bitkom@bitkom.org
www.bitkom.org