Effizientere Software-Entwicklung durch Industrialisierung der Prozesse

Oliver F. Nandico

oliver.f.nandico@capgemini.com

Zusammenfassung

Es gibt heute keine Branche, kein Unternehmen und keine Geschäftsprozesse, bei denen nicht „ein Stück Software“ involviert ist. Wir können uns eine ganze Reihe alter und vor allem neuer Geschäftsmodelle ohne IT-Unterstützung gar nicht mehr vorstellen.

Trotz, oder gerade aufgrund dieser Alltäglichkeit befindet sich die Entwicklung von Software in einem grundsätzlichen Wandel als Reaktion auf widersprüchliche Anforderungen. Verfallende Marktpreise und steigender Produktivitätsdruck verlangen die Industrialisierung der Entwicklungsprozesse. Das bedeutet Wiederholbarkeit, Standardisierung und Automatisierung. Gleichzeitig fordern die Anwender von Software die schnelle, innovative und ganz spezifische Reaktion des Softwareentwicklers auf ihre besondere Aufgabenstellung und ihr Geschäftsmodell. So erzielen sie ihrerseits einen Vorsprung im Wettbewerb vor der Konkurrenz.

Softwareentwicklung muss also heute Standardisierung und Automatisierung geeignet mit Flexibilität, Reaktionsfähigkeit und Nutzernähe kombinieren.

Diese Veränderung in der Softwareentwicklung wirkt sich konsequenterweise auf die Tätigkeit und das Berufsbild des Software-Ingenieurs aus. Zu diesem Wandel formuliert dieser Artikel sieben Thesen.

These 1: Standardisierter Prozess

These 1: Software-Entwicklung findet heute in einem wiederholbaren, standardisierten und arbeitsteiligen Prozess statt.

Mit „The greatest improvements in the productive powers of labour, and the greater part of the skill, dexterity, and judgment, with which it is anywhere directed, or applied, seem to have been the effects of the division of labour.“ beginnt schon Adam Smith sein Werk „Wealth of Nations“.

Voraussetzung für eine produktivitätssteigernde Arbeitsteilung ist ein klar definierter Prozess, in dem die Aufgaben und die Tätigkeiten für jede Rolle, und Vor- und Nachbedingungen festgelegt sind. Unter dem heute herrschenden Produktivitätsdruck in der Softwareentwicklung ist damit die arbeitsteilige Erstellung von Software in einem standardisierten Vorgehen Pflicht für jede IT-Abteilung und erst recht für jeden IT-Dienstleister.

Die industrialisierte Arbeitsweise hat den genialen Einzelprogrammierer oder das hervorragende Forschungsteam als Schöpfer neuer Programmierverfahren und Algorithmen in der täglichen Arbeit, im „Brot- und Butter“-Geschäft der Informatik abgelöst.

In der Konsequenz ersetzt der Spezialist für bestimmte Tätigkeiten im Entwicklungsprozess oder für bestimmte inhaltliche Fragestellungen den Generalisten. Diesen Verlust ganzheitlicher Handwerkskunst kann man bedauern, aber er ist ein Zeichen für die gewachsene Reife unserer Branche. Nicht zuletzt macht ein standardisierter und arbeitsteiliger Prozess die Softwareentwicklung plan- und kalkulierbarer, damit letztendlich erst erfolgreich.

Handwerklicher Vorgehensweise sind zudem natürliche Grenzen im Hinblick auf Komplexität und Größe eines Vorhabens gesetzt. Gerade die fast vollständige Durchdringung aller Funktionsbereiche von Unternehmen mit Software macht die Erstellung von neuen Programmen heute so schwierig und umfangreich, dass eine arbeitsteilige Vorgehensweise geboten ist.

Der arbeitsteilige und standardisierte Entwicklungsprozess bringt einen weiteren Vorteil: Er ist mess- und quantifizierbar. Erst wenn etwas wirklich messbar ist, kann man es optimieren, klar planen und scharf kalkulieren. Dies wiederum ist unter den verfallenden Marktpreisen für jedes Entwicklungshaus überlebensnotwendig.

Welcher Prozess ist aber der Richtige? Der Rational Unified Process ist sicher weit verbreitet und ein gewisser Standard, nicht zuletzt bei Capgemini.

Im deutschen öffentlichen Dienst ist das V-Modell XT als Standard vorgegeben. COBIT und ITIL sind gute Referenzmodelle für die Softwareentwicklung. Natürlich praktizieren eine Reihe von Entwicklungsteams, bei uns und anderswo, und ganze Firmen agile Vorgehensweisen. Wichtig ist: Der definierte Prozess muss der Komplexität und Aufgabe angemessen, und damit entsprechend konfigurierbar sein. Eine Customer-Relationship-Management-Lösung für ein globales Unternehmen ist anders zu entwickeln als eine Android-Sales-App.

Alle starren, definierten Prozesse sind entsprechend zu verwerfen, andere Anforderungen betrachten wir in der Diskussion der weiteren Thesen.

Was verlangt nun die Einführung standardisierter Prozesse von einem angehenden Softwareingenieur? Zum einen muss er die Diskussion um den Softwareentwicklungsprozess kennen, und gestalten können. Softwaredienstleister wie etwa unser Haus erwarten von ihren angehenden Beschäftigten Wissen um Aktivitäten und Zusammenhänge in der Softwareentwicklung. Der Softwareingenieur muss nach seiner Ausbildung wissen, was die Branche unter bestimmten Arbeitsergebnissen versteht, welche Anforderungen an diese sein Team und der Kunde stellt und über welche Aktivitäten, Methoden und Techniken sie zustande kommen. Kurzum der angehende Softwareingenieur braucht eine Ausbildung in den „industriellen Fertigungsprozessen“ der Softwareentwicklung.

These 2: Verringerte Wertschöpfung

These 2: Die Wertschöpfungstiefe in der Entwicklung verringert sich

“There’s only really one metric to me for future software development, which is — do you write less code to get the same thing done?” sagte Bill Gates 2005 im Hinblick auf die Veränderungen in der Software-Entwicklung.

Weniger Code bedeutet eine Verringerung Wertschöpfungstiefe in der eigentlichen Entwicklung. Diese Tendenz besteht seit Jahrzehnten und hat sich immer weiter verstärkt. Das führte zur Nutzung von ausgefeilteren technischen Produkten wie Transaktionssteuerungen, Datenbankmanagementsystemen und Bibliotheken für fast jeden Anwendungszweck. Schließlich zu immer „höheren“ Programmiersprachen.

Diese Frage der Wertschöpfung diskutieren die Anwender immer wieder unter der dichotomischen Sichtweise von „Standard-Software“, also (Halb-) Fertigsoftware, und Individualsoftware, „make or buy“. Die eine Seite führt immer Preis, Erstellungs- und Wartungsaufwand, die andere Seite die optimale Unterstützung der spezifischen Geschäftsprozesse und den differenzierenden Charakter als wesentliche Entscheidungskriterien an.

Dieser hebt sich aber zunehmend auf: Längst sind Softwarepakete komplexe, konfigurierbare und damit an die jeweiligen spezifischen Anforderungen anpassbare Systeme, denen im Übrigen der Kostenvorteil zunehmend abhandenkommt. Individualentwicklung setzt dagegen immer stärker und nicht nur als Lippenbekenntnis auf Wiederverwendung von bestehenden (Teil-) Lösungen, auf domänenspezifische Sprachen und Programmierung.

Längst gibt es darüber hinausweisend die Angebote „aus der Wolke“. Salesforce.com überzeugt seine Kunden seit mehr als 10 Jahren, dass sie überhaupt nichts mehr mit Softwareentwicklung zu tun haben müssen. Was bedeutet das für die interne IT-Abteilung?

Die Softwareentwicklung ist damit in Zukunft in der überwiegenden Breite mehr das Zusammenfügen von großen und kleinen Bausteinen zu einer dem Nutzer optimal angepassten Lösung. Integration von Software aus ganz unterschiedlichen Quellen bekommt so eine viel wichtigere Rolle als die Programmierung von Funktionen. Es gilt das Grundprinzip der Wiederverwendung: Schon wenige Tage Recherche nach einer angebotenen Lösung können Monate an Entwicklungsarbeit ersetzen.

Das Paradigma für den Umgang mit dieser Entwicklung ist die serviceorientierte Architektur: Der Service, die angebotene, in sich abgeschlossene und elementare fachliche Leistung ist der Baustein, um den sich die Softwareentwicklung dreht. Softwarepakete müssen solche integrierbaren Services anbieten, die Individualentwicklung erstellt sie und Cloud-Betreiber stellen sie über das Web zur Verfügung. Entwicklung wird damit mehr und mehr zur Architekturarbeit, zur richtigen Orchestrierung von Services.

Der Prozess der Softwareentwicklung muss entsprechend dieser Entwicklung serviceorientierte Architektur unterstützen. Das heißt er muss insbesondere Integration und die Schaffung einer Integrationsplattform angemessen berücksichtigen. Im Weiteren unterscheidet dann der Prozess der Softwareentwicklung nicht mehr zwischen individualentwickelten oder fertig bereitgestellten Services.

Bedeutet es nun, dass wir für die Ausbildung in der Softwareentwicklung die intensive Beschäftigung mit den Produkten eines Herstellers erwarten? Sicherlich nicht, denn alles Spezialwissen ist am Ende der Ausbildung mit hoher Wahrscheinlichkeit schon wieder völlig veraltet. Wichtig sind dagegen Fertigkeiten zum Umgang mit Softwarepaketen, zur Integration von Softwarepaketen aus verschiedenen Quellen mit individuell entwickelter Software. Klar formuliert: Nicht der XY-

Modulexperte muss Ziel der Ausbildung sein, sondern der Architekt von Anwendungslandschaften. Dabei gilt natürlich: Dieser Architekt kennt die am Markt angebotenen Produkte, ihre Stärken und Schwächen.

These 3: Wachsende Automatisierung

These 3: Der Automatisierungsgrad in der Softwareentwicklung muss steigen

„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.“ So beschreibt der „Leitfaden und die Orientierungshilfe“ der BITKOM zum Thema „Industrielle Softwareentwicklung“ den Aspekt Automatisierung.

In der Fertigungsindustrie ist eine Vorgabe für die Produktivitätssteigerung und entsprechende Rationalisierung von 10-15% im Jahr völlig normal und allgemein üblich. Will die Softwarebranche diesem Beispiel folgen, ist vor allem eine Automatisierung in der Entwicklung notwendig, neben der Standardisierung des Prozesses und der Verringerung der Wertschöpfungstiefe.

Automatisierung in der Softwareerstellung bedeutet die möglichst formale, konsistente und vollständige Beschreibung der Aufgabenstellung auf möglichst hohem semantischem Niveau, um alles weitere daraus automatisch zu generieren.

Dahinter steht die schon immer verfolgte Absicht, die tatsächliche Programmierarbeit in ihrer Komplexität und Fehleranfälligkeit zu vermeiden. Diese Absicht leitete schon die Entwicklung der ersten Compiler.

Heute bedeutet Automatisierung vor allem Einsatz von modellgetriebenen Techniken. Grundlage ist eine von der technischen Implementierung unabhängige, im Regelfall grafische Spezifikation der Software-Lösung. Entsprechende Werkzeuge transformieren diese in mehreren Schritten zur ablauffähigen Implementierung.

Teil der Automatisierung ist die Automatisierung für den Test von Software. Regressionsfähigkeit von Test und angemessener Abdeckungsgrad lassen sich überhaupt nur über automatisierte Tests erreichen. Hier ist der allgemein anzutreffende Reifegrad bei den Anwenderunternehmen eher niedrig.

Modellgetriebene Softwareentwicklung und Testautomatisierung sind heute in vielen Bereichen der Softwareentwicklung Standard, die Werkzeugwelt fast unüberschaubar. Dennoch zeigen sich noch Schwachstellen: Durchsatzoptimierung verlangt Anpassungen an die konkrete Hardware-Plattform. Daraus resultierende manuelle Eingriffe in den automatisch erzeugten Code zu erhalten, bedeutet immer noch eine Herausforderung für die Werkzeuge.

Modellgetriebene Entwicklung ist heute damit ganz selbstverständlich Teil des Softwareentwicklungsprozesses, Codegenerierung ist Stand der Technik. Diese Entwicklung wird und muss sich fortsetzen, im Sinn einer weiteren Industrialisierung. Das schränkt die manuellen Codierungsarbeiten ein, und verändert so sicher den Beruf des Software-Ingenieurs weiter.

In der Konsequenz muss die Ausbildung für Softwarenentwicklung den Umgang und Einsatz mit Werkzeugen zur Automatisierung in jeder Beziehung lehren. Modellgetriebene Softwareentwicklung ist in unserer Branche ebenso wichtig, wie der Umgang mit Fertigungsrobotern und CNC-Maschinen im Maschinenbau.

These 4: Geografische Verteilung

These 4: Software-Entwicklung erfolgt geografisch verteilt

„It’s the combination of flexibility, the right competencies, the blend of onshore and offshore resources and cost savings that make Rightshore® such an attractive proposition.” Stefan Fransson, CIO, Mölnlycke Health Care

Es ist kein Geheimnis, dass die Verlagerung von Softwareentwicklungsarbeiten in Regionen mit geringeren Lohnkosten internen IT-Abteilungen erhebliche Kostenvorteile bringen und IT-Dienstleistern niedrigere Preise ermöglichen. Dies ist, trotz der zunehmenden Automatisierung, dem sehr hohen Lohnkostenanteil von 70% (Schweizerische Technische Zeitschrift) in der Softwareentwicklung geschuldet.

Dazu fallen für Software keine im eigentlichen Sinn Logistikkosten an. Natürlich muss in den entsprechenden Ländern die Infrastrukturanbindung gewährleistet sein, also Internetverbindung existieren. Hier leistet etwa in Indien der Staat hohe Vorleistungen. Das gilt analog für die Ausbildung der Entwicklerinnen und Entwickler.

Viele Kunden haben die Erfahrung gemacht, dass eine einfache Verlagerung der Arbeiten in Standorte mit niedrigeren Löhnen nicht zu den gewünschten Ergebnissen führt. Das fachliche Verständnis war häufig nicht gegeben, Kommunikationsprobleme haben die die Fertigstellung der Software verzögert und den Aufwand dafür erhöht.

Geografische Verteilung der Softwareentwicklung verlangt einen sehr angepassten Entwicklungsprozess, in dem Aufgaben und die Interaktion zwischen den Prozessbeteiligten in ihren Rollen sehr sorgfältig definiert sein müssen.

Die Arbeit in einem räumlich, d.h. geografisch verteilten Team verlangt einen wohldefinierten und arbeitsteiligen Prozess in der Entwicklung, der allen Beteiligten hinreichend klar ist.

Von den Beteiligten in der Software-Entwicklung erfordert er zusätzliche soziale Kompetenzen, vor allem interkulturelle Fähigkeiten, die über reine Sprachkenntnisse hinausgehen.

Genau dieser Aspekt ist in der Ausbildung zu verstärken. Das sind dann Sprachkurse, das sind mehr Auslandspraktika. Interkulturelle Fähigkeiten kann man nicht aus dem Buch lernen, man muss sie erleben. Die Erwartungen aus dem Bologna-Prozess harren allerdings leider noch ihrer Erfüllung.

These 5: Mehr Kundennähe

These 5: Die Nähe zum Kunden und die Vertrautheit mit seiner Aufgabenstellung müssen sich erhöhen

„We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: Individuals and interactions over processes and tools, Working software over comprehensive documentation, Customer collaboration over contract negotiation, Responding to change over following a plan.“ steht im „Agilen Manifest“ von 2001.

Während die bisherigen Thesen alle auf einen wohldefinierten, detaillierten, eben industrialisierten Softwareentwicklungsprozess weisen, zeigt die Diskussion und der Erfolg agilen Vorgehens, dass Kundennähe und die schnelle Reaktion auf veränderte Anforderungen unverzichtbar für den Erfolg eines Softwareprojektes sind.

Vorgehensmodelle, die ein Vorhaben nötigen an einer einmal getroffenen fachlichen Entscheidung oder an dem am Beginn festgelegten funktionalen Umfang festzuhalten, sind zum Scheitern verurteilt. Aus diesem Grund hat niemand Wasserfallmodelle, bei ehrlicher Betrachtung des tatsächlichen Vorgehens, wirklich praktiziert. Lange Spezifikationsphasen und bei Fertigstellung veraltete Software waren und sind verbreitete Abbruchsgründe.

Der Kern für den Erfolg von agilen Methoden ist die frühe und konsequente Einbeziehung des Kunden und Anwenders. Dazu kommt die Konzentration auf lauffähige Software, um ihm die Beurteilung und Steuerung des Projektes zu ermöglichen.

Grundsätzlich bedeutet Agilität nicht wie bisher die Abwehr aller „störenden“ Änderungsanforderungen, sondern die bewusste Ausnutzung der Flexibilität der Softwareentwicklung, um den Nutzen für den Kunden zu maximieren. Im schlechtesten Fall erhält der Kunde am Schluss ein System mit der Hälfte des gewünschten Funktionsumfangs, das aber dennoch lauffähig ist. Das ist besser als eine vollständige Spezifikation, mit der er überhaupt nichts anfangen kann.

Es gibt jedoch Gefahren beim agilen Vorgehen. Das Entwicklungsteam kann Details übersehen. Lokale und kurzfristige Optimierungen können umfassende und nachhaltig tragfähige Gesamtlösungen verhindern. Entscheidungsschwache oder fachlich inkompetente Kundenansprechpartner verhindern den Erfolg.

Mit agilem Vorgehen sind auch Missverständnisse verbunden. Agilität heißt nicht ungeordnetes, auf bloßer direkter Kommunikation basierendes Vorgehen und Verzicht auf jegliche Dokumentation. Vielmehr verlangen agile Prozesse, wie zum Beispiel SCRUM, weitaus mehr Wissen und Erfahrung mit dem Prozess und mehr Disziplin von allen Beteiligten.

Eine Kontroverse besteht, inwieweit agile Vorgehensweisen bezüglich Projektgröße und Projektkomplexität skalieren. Erfahrungen aus unserem Haus deuten in die Richtung, dass wir im Einsatz von agilen Techniken sehr genau nach Projekttyp unterscheiden müssen. Das heißt, dass agile Techniken dort am besten wirken, wo in einem eher kleinen Team in kurzer Zeit hoch innovative Lösungen entstehen sollen. Dessen ungeachtet können agile Vorgehensweise die Effektivität und Effizienz bei großen und umfangreichen Lösungen erhöhen. Hier befinden wir uns noch in einer Phase der Diskussion und Forschung.

Eine andere Frage stellt sich: Interessiert es den Kunden überhaupt, nach welchem Prozess ein Dienstleister für ihn die Software entwickelt? Die Erfahrung bestätigt das. Wenn, wie in unserer Branche ein Übergangsstadium zur Industrialisierung gegeben ist, und gleichzeitig der Anteil von gescheiterten Entwicklungsprojekten nach wie vor sehr hoch ist, dann ist das Vorgehen und damit der Prozess ein wesentliches Argument in der Auftragsvergabe. Zusicherungen in Bezug auf zeitverkürzter Abwicklung und die Anforderung nach intensiver Mitwirkung des Kunden im agilen Prozess verlangen die Darstellung von Prozess und Vorgehen. Folgerichtig gibt es ein sehr vitales Interesse des Kunden in Bezug auf die Umsetzung entsprechend dem zugesicherten Verfahren.

Für die Ausbildung gilt hier eine einfache Anforderung: Kenntnisse und Erfahrung in agilen Techniken. Erfahrung heißt hier, ein Projekt, also etwa ein Praktikum nach einem solchen Vorgehen im Team praktiziert zu haben. Das stellt natürlich entsprechende Anforderungen an die Ausbilder.

These 6: Schneller!

These 6: Umsetzungsgeschwindigkeit ist essentiell

„Während ein Projektleiter vor einiger Zeit noch stolz sein konnte, wenn er nach einem Jahr ein fertiges System abliefern konnte, muss er heute diese Aufgabe in längstens einem halben Jahr erledigen können“ (CIO eines Kundenunternehmens, 2007).

Aller Produktion, und das schließt Softwareentwicklung ein, ist gemeinsam, dass die Anwender eine Verringerung der Prozessdurchlaufzeiten fordern. Eine wirtschaftliche Chance hat typischerweise ein begrenztes Zeitfenster, ein Wettbewerbsvorsprung besteht nur in einem engen zeitlichen Rahmen. Wenn Software dafür notwendig ist, und das ist fast immer der Fall, dann muss sie **jetzt** zur Verfügung stehen.
Standardisierung, Automatisierung, selbst die Qualität der Software steht aus Sicht der Anwender hinter der Ausnutzung des Chancenfensters zurück.

Diese grundsätzliche Anforderung hat es schon immer gegeben. Darauf hat das Softwareprojekt mit Stufenkonzepten und Systemdurchstichen reagiert. Die Erwartungshaltung der Anwender hat sich über die Zeit verstärkt: Wo diese Anwender die Entwicklungszeiten für neue Automodelle oder auch neue Flugzeugtypen reduzieren müssen, gibt es kein Verständnis, wenn die Software, die damit einhergeht, nicht diese Zeitvorgabe schafft.

Unter dem gegebenen zeitlichen Druck sind die Kunden häufig bereit, Ansprüche an die Qualität der Software zurückzustellen. Das führt langfristig zu schlechten Lösungen, die weder anpassbar noch betreibbar sind. Hier ist es die Aufgabe der Softwareentwicklung, über die Gestaltung des Prozesses eine Mindestqualität von vornherein zu sichern. Dafür gibt es entsprechende Techniken, die ebenfalls ihre Wurzeln in der Diskussion um agile Techniken haben. Beispielhaft seien hier nightly Builds, frühe Unit Tests und Prototypen genannt.

Trotz der Ausrichtung auf den wohldefinierten Prozess sind Umfang, sowie Art und Weise der Dokumentation einer kritischen Betrachtung zu unterziehen. Dokumente, die keine Wirkung haben und keine Entscheidung beeinflussen, benötigt auch niemand. Für jedes im Prozess zu liefernde Zwischenergebnis muss am Beginn eines Entwicklungsprojektes die Projektleitung Sinn und Zweck bestimmen können. Ergebnisdokumente, deren einziger Zweck die Erfüllung einer Anforderung aus der Prozessdefinition ist, halten die Entwicklung nur unnötig auf.

Zusätzlich zu den Anforderungen an die Ausbildung aus der Diskussion der anderen Thesen ergibt sich hier nur noch einmal hervorzuheben: In der Ausbildung ist nicht nur Lösungs- sondern zukünftig und vor allem Prozesskompetenz gefordert.

These 7: Angemessen!

7: Angepasste Prozesse sind der Schlüssel zum Erfolg

„The danger of standard process is that people will miss chances to take important shortcuts." Dieses Zitat von Tom DeMarco zeigt schon richtig eine der wesentlichen Gefahren eines wohldefinierten Entwicklungsprozesses auf: Er ist nicht an die eigentliche Aufgabe angepasst.

Viele Vorgehensmodelle in der Vergangenheit, wie etwa das V-Modell, und verschiedene Ausprägungen von Vorgehensmodellen bei verschiedenen Firmen haben oft eine Gemeinsamkeit: Sie sind relativ starr, ihre Dokumentation ist komplex und umfangreich und kein betroffener Softwareentwickler hat sie angewendet oder auch nur verstanden.

Das hängt damit zusammen, dass Theoretiker in verschiedenen Definitionen der Prozesse versucht haben, auf der abstrakten Ebene jedes Entwicklungsproblem zu lösen oder zumindest den Lösungsweg über Aktivitäten und Ergebnisdokumente vorzugeben. Das hilft den Entwicklern nicht und macht die Beschreibungen nur unverständlich.

Die Idee, für die ganz unterschiedlichen Aufgabenstellungen in ganz unterschiedlichen Softwareprojekten den gleichen Prozess aufzusetzen, ist zum Scheitern verurteilt. Das Softwareprojekt, das in einem engen vorgegebenen Zeitrahmen ein Preisvergleichsportal erstellt, hat mit einem Projekt für die Entwicklung der Steuerungssoftware für einen Laborautomaten nur wenige Gemeinsamkeiten.

Das Angebot, einen wohldefinierten Standardprozess zu konfigurieren, hilft dann nur bedingt weiter, wenn der Ausgangspunkt zu komplex und abstrakt ist.

Es gibt, wie die Diskussion der Thesen gezeigt hat, verschiedene Aspekte, die auf unterschiedliche Entwicklungsprozesse hindeuten. In unserem Hause sehen wir sechs Dimensionen, die für den angemessenen Entwicklungsprozess relevant sind:

- Priorisierung zwischen Funktionsumfang und Zeit
- Änderungsdynamik bei den Anforderungen
- Kundenkultur
- Einbindung des fachlichen Auftraggebers
- Grad der Abhängigkeiten zwischen einzelnen Teilkomponenten und –funktionen
- Projektgröße

Aus diesen Dimensionen ergeben sich die Aspekte für den jeweils angemessenen Entwicklungsprozess. Im Einzelnen heißt das insbesondere, dass der Grad an agiler Vorgehensweise und der Einsatz von bestimmten agilen Techniken, der Einsatz von

Offshore-Entwicklungskapazitäten und die Einbindung von Softwarepaketen bzw. Cloud-Angeboten daraus abzuleiten ist.

Daraus ergeben sich unterschiedliche Typen von Projekten und Entwicklungsprozessen. Eine nicht abschließende Liste ist:

- Entwicklung relativ kleiner, technisch wie fachlich hoch-innovativer Systeme, die in einem engen Zeitrahmen produktiv werden müssen.
- Entwicklung funktional umfangreicher, fachlich und architektonisch komplexer Systeme, die die differenzierenden Kernleistungen eines Unternehmens unterstützen.
- Integration von verschiedenen schon existierenden oder neu zu entwickelnden Services mit den Angeboten aus Softwarepaketen und Cloud-Angeboten.

Für so zu identifizierende Projekttypen gilt es nun für das Management der Softwareentwicklung, einfache und verständliche Prozessdefinitionen zu erstellen. Auf dem Weg zum industrialisierten Prozess ist nach der Definition dann der zweite Schritt die Herstellung der Automatisierung und Unterstützung der Entwickler in der Auswahl des angemessenen Prozesses, sowie der Konfiguration und Umsetzung dieses so ausgewählten Prozesses.

Die Konsequenzen für Software-Entwickler

Was bedeutet nun die Industrialisierung in der Softwareentwicklung für das Berufsbild des Software-Ingenieurs?

Zunächst einmal heißt es, dass die Reflektion des Entwicklungsprozesses zum selbstverständlichen Kanon in der Ausbildung gehört. Die Grundlage dieser Reflektion sind vertiefte Kenntnisse zu den Standardvorgehensmodellen. Hierzu gehört sicherlich der Rational Unified Process und das V-Modell XT. Dazu kommen die heute ganz standardmäßig vorausgesetzten Referenzprozessmodelle ITIL und COBIT. Eine Wunschvorstellung ist es, wenn Absolventen aus der Hochschulausbildung hier schon entsprechende Zertifizierungen mitbringen.

Genauso, wie statt einer bestimmten Programmiersprache die dahinter stehenden Konzepte wie Objekt- und Aspektorientierung gefragt sind, ist es für Vorgehensmodelle entscheidend, ihren Zweck und beabsichtigten Einsatzrahmen zu kennen und zu wissen, wo und wie sie anzupassen sind.

Die Softwareentwicklung hat einen hohen Bedarf an Prozessingenieuren und –ingenieurinnen. Diese brauchen wir für die Aufnahme, Modellierung und Optimierung von fachlichen Prozessen in den verschiedenen Branchen, nicht zuletzt für die eigene, die Softwareentwicklung.

Heute führen wir für unsere Beschäftigten interne Schulungen zu diesem Thema durch. Dazu leistet eine eigene Gruppe „Produktionssteuerung“ die notwendige Unterstützung der Projekte. Diese Unterstützung ist notwendig, weil wir in unserem Hause feste Vorgaben für die Abwicklung und Durchführung von Softwareentwicklungsprojekten machen. Die Weiterentwicklung von Methoden und Vorgehensmodellen ist schließlich in der Kompetenz unserer Entwicklungsabteilung. Insgesamt betreiben wir einen hohen Aufwand in der Umsetzung von definierten und standardisierten Prozessen. Wie schon beschrieben, sind für uns die ständige Verbesserung der Produktivität, und die schnelle Reaktion auf neuere Entwicklungen, die darauf beruhen, essentiell.

Gerade im Rahmen einer weiteren Industrialisierung bleiben für Softwareingenieure die sozialen Kompetenzen wichtig. An vorderster Stelle stehen dabei Teamfähigkeit und Kommunikationsfreude. Trotz aller Standardisierung und Automatisierung bleibt Softwareentwicklung ein kreativer Prozess, bei dem ein hochqualifiziertes Team in der Zusammenarbeit optimale Lösungen sucht und findet. Zunehmend wichtiger ist dabei die globale Ausrichtung, die interkulturelle Kompetenz über bloße Sprachkenntnisse hinaus verlangt.

Literatur

Agile Manifesto (2001) unter:
http:/ / agilemanifesto.org

BITKOM 2010, Industrielle Softwareentwicklung, unter:
http:/ / www.bitkom.org/ files/ documents/ Industrielle_Softwareentwicklung_web.pdf

DeMarco, T.; Lister, T. (1999): Peopleware: Productive Projects and Teams, Dorset House Publishing Company, 2nd edition, New York N.Y. 1999

Kremer, M. (2010): eee@Quasar: efficient, effective, and economic software development, Vortrag Capgemini Kundenforum Architektur

Smith, A. (2005): An Inquiry into the Nature and Causes of the Wealth of Nations, Pennsylvania State University, Hazelton 2005

Udell, J. (2005): Interview with Bill Gates, Infoworld 25. September 2005

Software made in India (2008)
in: Schweizerische Technische Zeitschrift – Swiss Engneering, Ausgabe November 2008, S.10-11