Situational Software Engineering: Ein Rahmenwerk für eine situationsgerechte Auswahl von Entwicklungsmethoden und Vorgehensmodellen

Gregor Engels[1,2], Marion Kremer[2]

[1] Universität Paderborn
s-lab – Software Quality Lab
Warburger Str. 100
33098 Paderborn
engels@upb.de

[2]Capgemini
CSD Research
Berliner Straße 76
63065 Offenbach
marion.kremer@capgemini.com

Abstract: Industrielle Softwareentwicklung unterliegt einem stetig steigenden Kosten-, Zeit- und Qualitätsdruck, der einen effektiven, effizienten und wirtschaftlichen Einsatz der zur Verfügung stehenden Ressourcen bei IT-Dienstleistern erfordert. In diesem Beitrag berichten wir über industrielle Erfahrungen bei der Entwicklung eines Rahmenwerks, um für ein aktuelles Softwareprojekt im Sinne eines Situational Software Engineering eine situationsgerechte Auswahl und Kombination von Methoden- und Vorgehensmodellbausteinen zu bestimmen. Es werden konkrete Kriterien angegeben, anhand deren Methodenbündel für Systemklassen geschnürt werden können.

1 Einleitung und Motivation

Ein wesentliches Ziel der Software Engineering-Forschung besteht darin, Methoden und Vorgehensmodelle zu entwickeln, um die Softwareentwicklung so weit wie möglich zu industrialisieren. Hierdurch soll einerseits eine hohe Qualität der entstehenden Softwareprodukte gewährleistet werden, andererseits sollen und müssen die vorhandenen Ressourcen effizient, effektiv und kostenbewusst eingesetzt werden.

Aufgrund veränderter Rahmenbedingungen sind hierbei die Anforderungen an den Einsatz von Methoden und Vorgehensmodellen stark gestiegen. Beispiele für veränderte Rahmenbedingungen sind einerseits die immer häufiger geforderte Entwicklung von integrierten Anwendungslandschaften auf der Basis existierender oder extern entwickelter Basiskomponenten im Gegensatz zur Individualentwicklung einzelner Anwendungssysteme. Andererseits steigen die Einschränkungen bezüglich verfügbarer Ressourcen (Zeit, Budget, Fachpersonal) sowie Compliance-Anforderungen bezüglich zu verwendender Standards und Technologien.

Der IT-Dienstleister Capgemini (vormals sd&m) hat in den letzten 20 Jahren unter dem Namen Quasar 2.0 ein Methodenportfolio für die Softwareerstellung entwickelt, das insbesondere auf die Entwicklung von Individualsoftware (Custom Software Development (CSD)) zugeschnitten ist (siehe Abb. 1) [Cap09, Sie04]. Es ist strukturiert in Domänen (Requirements, Analysis & Design, …) sowie in konstruktive (hellblaue) und analytische (dunkelblaue) Disziplinen.

erschienen im Tagungsband der INFORMATIK 2011
Lecture Notes in Informatics, Band P192
ISBN 978-3-88579-286-4
weitere Artikel online:
http://informatik2011.de/519.html

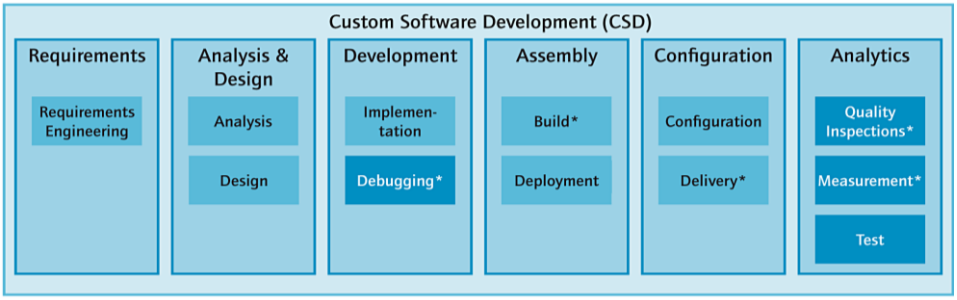

Abb. 1: Quasar 2.0

Quasar 2.0 orientiert sich an plangetriebenen Vorgehensmodellen wie wasserfallähnlichen, inkrementellen bzw. iterativen, an RUP (Rational Unified Process) angelehnten Vorgehensmodellen [JBR09]. Aufgrund von Markt- und Kundenanforderungen werden in einzelnen Projekten methodische Elemente von Quasar 2.0 auch mit einer agilen Vorgehensweise verknüpft. Um dies zu unterstützen wurde eine werkzeuggestützte Selektionsmethode erarbeitet [HE10], die Hilfestellung bei der Auswahl des passenden Vorgehensmodells liefert.

Der nächste Schritt in der Weiterentwicklung von Quasar ist nun im Sinne eines *Situational Software Engineering* eine verstandene Vorgehensweise, wann welches Vorgehensmodell mit welchen methodischen Bausteinen gekoppelt wird, um eine möglichst effiziente, effektive und Preis-effiziente, d.h. in Summe wirtschaftliche Vorgehensweise für ein aktuelles Projekt zu haben. Aus diesem Grunde wird zurzeit an einem Rahmenwerk für eine situationsgerechte Auswahl von methodischen Bausteinen und Vorgehensmodellen gearbeitet, auf dessen Basis ein Selektionswerkzeug für die Festlegung einer optimalen Kombination von Methodenbausteinen und Vorgehensweisen entwickelt werden kann. Der aktuelle Stand der Überlegungen zu diesem Rahmenwerk ist Inhalt und Ziel dieses Beitrags.

Grundsätzlich steht dieses Vorgehen natürlich unter dem Vorbehalt, dass es die Vertragslage mit dem Kunden überhaupt zulässt, die Entwicklungsmethodik und das Vorgehensmodell festzulegen. Daher geht der vorliegende Beitrag davon aus, dass zu Entwicklungsmethodik und Vorgehensmodell vom Kunden keine einschränkenden Vorgaben gemacht werden. In den Fällen, in denen dies nicht zutrifft, ist das hier vorgestellte Verfahren geeignet anzupassen.

Der Beitrag ist wie folgt strukturiert. Im Kapitel 2 werden relevante Arbeiten aus dem Bereich des Prozess- und Methoden-Engineering dargestellt, deren Erkenntnisse in die Entwicklung des Rahmenwerks einfließen. Im Kapitel 3 stellen wir dar, was wirtschaftliche Entwicklung in diesem Zusammenhang bedeutet. Anschließend werden in den Kapiteln 4 und 5 der aktuelle Stand der Diskussion zu Entwicklung und Einsatz des Rahmenwerks vorgestellt. Kapitel 6 beschließt den Beitrag mit einer Zusammenfassung und einem Ausblick auf zukünftige Arbeiten.

2 Prozess- und Methoden-Engineering

Vorgehensmodelle dienen bei der Softwareentwicklung dazu, den Prozess der Softwareentwicklung zu strukturieren, d.h. insbesondere die durchzuführenden Aktivitäten und ihre möglichen Reihenfolgen, die den Aktivitäten zugeordneten Verantwortlichen und die durch die Aktivitäten bearbeiteten bzw. erstellten Dokumente (bzw. Artefakte) festzulegen. Die eigentlichen zur Anwendung kommenden Methodiken für die Durchführung der Aktivitäten wie z.B. eine Spezifikationsmethodik sind dabei in Domänen (analog RUP) zusammengefasst. Sie enthalten übergreifende Beschreibungen, Richtlinien, Vorlagen, Best Practices, Werkzeuge usw. Dabei gliedern sich die Domänen in Disziplinen.

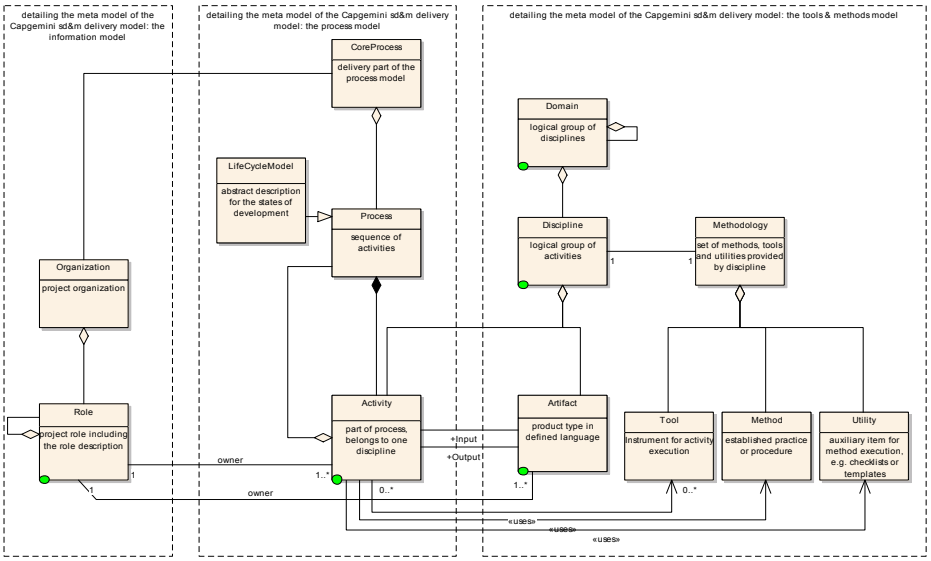

Abb. 2: Prozess- und Methoden-Meta-Modell bei Capgemini

Abb. 2 verdeutlicht in Form eines Metamodells die wesentlichen Bestandteile von Vorgehensmodellen und Methodenbausteinen. Es beruht auf Arbeiten bei Capgemini und beschreibt die grundlegenden Strukturen der Entwicklungsmethodik. Dies sind die organisationsbezogenen Elemente (linker Teil), die Prozess-bezogenen Elemente (mittlerer Teil) und die Methoden-bezogenen Elemente (rechter Teil). Eine derartige Metamodelldarstellung beruht auf Standardarbeiten im Bereich des Method und Process Engineering ([Bri96], [OMG08]).

In der Geschichte des Software Engineering wurden unterschiedliche Vorgehensmodelle entwickelt. Zu den bekanntesten gehören die sogenannten plangetriebenen wie das klassische Wasserfallmodell, der Rational Unified Process und das insbesondere in Deutschland bekannte V-Modell XT [RB07]. In der letzten Zeit haben daneben agile Vorgehensmodelle wie Scrum, Kanban oder XP eine steigende Bedeutung erreicht.

Kein Vorgehensmodell und keine Methode ist für jede Projektsituation gleich gut geeignet. Das bedeutet, dass für die jeweilige Projektsituation die ideale Kombination von Vorgehensweise und methodischen Bausteinen ausgewählt werden muss. Dies wurde auch bereits früh im Bereich Method Engineering erkannt und der Name „Situational Method Engineering“ geprägt [WK92]. Auch im V-Modell XT (eXtreme Tailoring) ist diese Erkenntnis bereits im Namen verankert. Hierbei wird unter Tailoring verstanden, einen von drei vordefinierten Projekttypen auszuwählen, die anzuwendenden Vorgehensbausteine festzulegen und eine Projektdurchführungsstrategie festzulegen.

Die Diskussion zwischen plan-getriebener und agiler Vorgehensweise wurde insbesondere von B. Boehm und R. Turner in [BT03] vorangebracht. Sie identifizieren eine Reihe von Schlüsselfaktoren, die hilfreich sind, um sich für eine der beiden Vorgehensweisen zu entscheiden. Dies sind (1) Größe des Softwareprodukts bzw. des Teams, (2) Kritikalität des Softwareprodukts im Einsatzkontext, (3) Qualität und Erfahrenheit der Entwickler sowie (4) Dynamik der Anforderungen.

Auf dieser Basis wurde auch bei Capgemini ein Selektionswerkzeug für die Auswahl eines Vorgehensmodells entwickelt [HE10]. Hierbei wurden als weitere Schlüsselfaktoren insbesondere der Architekturaspekt der Abhängigkeiten zwischen Features und Systemkomponenten (e) sowie der zeitliche Aspekt (a) hinzugenommen (siehe Abb. 3). Das Selektionswerkzeug unterstützt den Projektmanager bei der Zuordnung seines Projektes zu einem geeigneten Vorgehensmodell gemäß den in Abb. 3 angegebenen Faktoren (agil = dunkelrot, plangetrieben = dunkelblau). Falls das Projekt nicht eindeutig einem plangetriebenen oder agilen Vorgehensmodell zugeordnet werden kann (d.h. Projektcharakteristika liegen im hellblauen bzw. hellroten Bereich), wird vom Werkzeug eine Mischform empfohlen, wobei die dabei entstehenden Risiken dem Projektmanager angezeigt werden (näheres in [HE10]).

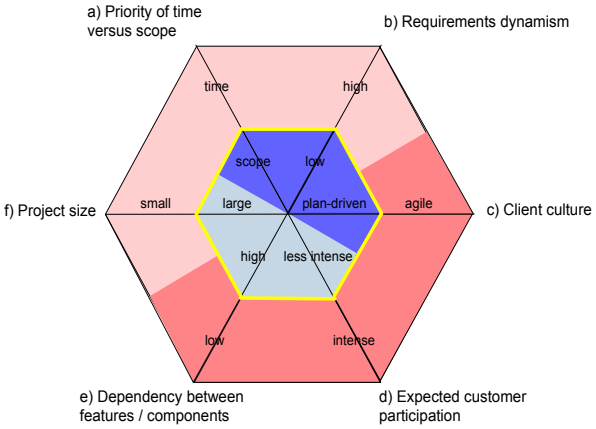

Abb. 3: Schlüsselfaktoren der Vorgehensmodell-Selektionsmethode bei Capgemini

Erfahrungen mit dem Selektionswerkzeug zeigen, dass Projektmanager nun eine bewusstere Entscheidung bei der Auswahl eines Vorgehensmodells treffen. Gleichzeitig zeigen die Erfahrungen im Einsatz des Selektionswerkzeugs, dass sie weitergehende Unterstützung bei der Frage benötigen, welcher Methodenbaustein in welcher Domäne Verwendung finden soll. Damit wird eine Erweiterung des Selektionswerkzeuges benötigt, dessen grundlegendes Rahmenwerk im Folgenden vorgestellt wird.

3 Wirtschaftliche Softwareentwicklung

Um als IT-Dienstleister am Markt erfolgreich zu sein, muss in jedem einzelnen Softwareentwicklungsprojekt eine situationsbezogene Kombination aus Vorgehensmodell und Methodenbausteinen gefunden werden, die insbesondere eine hohe Wirtschaftlichkeit der Softwareentwicklung garantiert. Hierbei beeinflussen die folgenden drei, im Konflikt stehenden Faktoren die Wirtschaftlichkeit (siehe Abb. 4)

- Effektivität: das Richtige wird getan (d.h. es wird nichts Unnötiges geleistet)
- Effizienz: das was getan wird, wird richtig getan (d.h. ohne Verschwendung von Ressourcen)
- Preis-Effizienz: Es werden die Ressourcen so eingesetzt, dass das Produkt aus Stückkosten und Aufwand möglichst gering ist (d.h. insbesondere auch, dass es sich durchaus lohnen kann, ein teureres Werkzeug einzusetzen, wenn damit der durch Menschen zu leistende Aufwand deutlich sinkt)

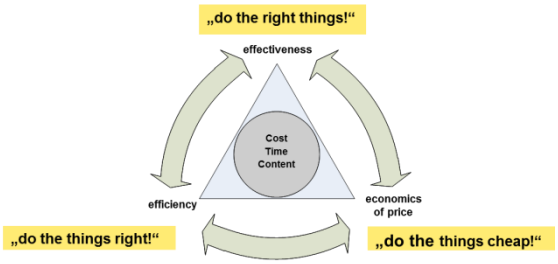

Abb. 4: Wirtschaftlichkeit im Gleichgewicht zwischen Effektivität, Effizienz und Preis-Effizienz

Da keine Optimierung in alle drei Richtungen gleichzeitig möglich ist, ist bei jedem Projekt sicherzustellen, dass die Balance zwischen diesen drei Faktoren zielgerichtet gesteuert wird. Ein Beispiel hierfür ist die Architekturentwicklung. Eine gute Architektur ist zwingend notwendig, da sie u.a. einen geringeren Implementierungsaufwand bedingt, aber andererseits Mehr-Aufwand beim Entwurf bedeutet. Gleichzeitig bringen typischerweise die letzten kleinen Optimierungen keine große Einsparung. Damit muss hier die Balance gefunden werden zwischen Effektivität und Effizienz.

Dieses Zusammenspiel zwischen Effizienz, Effektivität und Preis-Effizienz („economics of price“) spiegelt sich auch im neuen Titel „*eee@Quasar*“ der situationsbedingten Entwicklungsmethodik bei Capgemini wider.

ISBN 978-3-88579-286-4

4 Entwicklung des Rahmenwerks

Im Folgenden geben wir zunächst einen Überblick über die einzelnen Schritte zur Entwicklung des Rahmenwerks (vgl. Abb. 5), bevor wir die einzelnen Schritte erläutern.

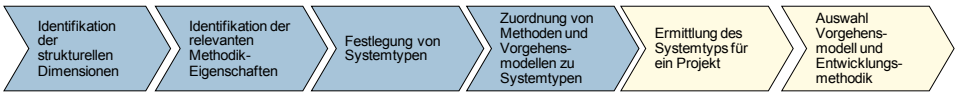

Abb. 5: Schritte zur Entwicklung des Rahmenwerks zur Auswahl von Vorgehensmodell und Methodenbausteinen

Bei der Entwicklung des Rahmenwerkes wurde deutlich, dass die eigentliche Schwierigkeit darin besteht, die Kriterien bzw. strukturellen Dimensionen sowie ihre Ausprägungen zu finden, die für eine Entscheidung bzgl. der Auswahl der Methodenbausteine herangezogen werden sollen. Insbesondere wurde hierbei deutlich, dass die im Laufe eines Entwicklungsprozesses entstehenden Artefakte nicht nur an dem Nutzen für eben diesen Entwicklungsprozess gemessen werden dürfen, sondern auch an den Erfordernissen der späteren Wartung und des Betriebs. Daher betrachten wir in der Folge statt Entwicklungsprojekten Systeme und deren Umfeld.

Ein Beispiel für eine derartige strukturelle Dimension ist die Lebensdauer mit den beiden möglichen Ausprägungen

- keine Wartungsphase mit neuen fachlichen Anforderungen (Beispiel dafür wäre ein kurzlebiges Online-Gewinnspiel, das nach dem entsprechenden Ereignis, wie der Fußball-WM, wieder abgeschaltet wird)
- sehr lange Wartungsphase (mehrere Jahrzehnte) mit zu erwartenden fachlichen Änderungen

Parallel zu der Erstellung dieses multi-dimensionalen Beschreibungsrahmens lassen sich Ausprägungen von Entwicklungsmethodiken identifizieren. Dabei sind bei der Auswahl bzw. Definition der konkreten Methoden alle drei Faktoren Effizienz, Effektivität und Preis-Effizienz zu berücksichtigen.

Spätestens an dieser Stelle ist das hier vorgestellte Rahmenwerk als Lösungsmuster zu verstehen, weil andere Gruppierungen innerhalb des Beschreibungsrahmens durchaus sinnhaft sein können, wenn der jeweilige Betrachtungsfokus ein anderer ist.

Diese Ausprägungen der Entwicklungsmethodik lassen sich dann verwenden, um Systemgruppen zu bilden, die dadurch gekennzeichnet sind, dass sie ähnliche Anforderungen an die Entwicklungsmethodik stellen.

Auf der Basis des so erstellten Rahmenwerks kann für jedes individuelle Projekt entschieden werden, wie es zu den Referenzmodellen steht und welche Kombination aus Vorgehensmodell und Methodikbausteinen ausgewählt werden soll.

ISBN 978-3-88579-286-4

Im Folgenden werden wir die einzelnen Schritte und die von uns gewählten Ausprägungen genauer darstellen.

Schritt 1: Identifikation der strukturellen Dimensionen eines Software-Entwicklungsprojektes

Für Capgemini als Anbieter kompletter IT-Lösungen spielt eine Vielzahl von strukturellen Dimensionen bei der Auswahl einer passenden Entwicklungsmethodik eine Rolle. Sie alle sind sehr wesentlich für die Projekte, weil sie deutliche Konsequenzen für den Entwicklungsprozess und die dafür zu verwendenden Methodikbausteine haben.

Die folgende Liste stellt die aktuell diskutierten strukturellen Dimensionen und ihre Extrem-Ausprägungen zusammen:

Structural Dimension	**Extrema**
Business / Problem characteristics	
degree of substitution of an existing system (low=greenfield)	low - high
Requirements Dynamics	low - high
# Differentiating Business Services	low - high
Functional Complexity / Problem Size / # Number of services to be provided	low - high
User Interaction Complexity	low - high
Neighbouring Systems Complexity	low - high
Dependency between components / features	low - high
Safety Requirements	low - high
Lifetime	low - high
Environment / Context	
Client culture	agile – plan-driven
Expected Customer Participation	low - high
Possible degree of off-shore (low = all on-shore)	low – high
Possible Degree of outsourcing (low = inhouse)	low - high
Possible Acceptance of package-based solutions / open source	low - high
Priority of time vs. scope	time - scope
Sourcing concept	onsite - offshore
Skill-Level	low - high
Team changes over life time	low - high
Amount of necessary traceability of decisions	low - high
Operations	
Internal operations / cloud (multi-tenant)	internal - external
necessary nfa's (performance, reliability, etc.)	low - high
way of operations	centralized - distributed

ISBN 978-3-88579-286-4

Schritt 2: Identifikation der relevanten Methodik-Eigenschaften

Die im Folgenden beschriebenen Eigenschaften von Methodikbausteinen sind solche, die sich in der bisherigen Diskussion bei Capgemini als relevant gezeigt haben:

- Art des Requirements Engineering und der Qualitätssicherung:

  Im Fall eines Greenfield-Projektes, bei dem es keine Vorlage (= Altsystem) gibt und die Fachlichkeit noch gemeinsam mit dem Kunden zu erarbeiten ist, ist das Requirements Engineering methodisch anders durchzuführen als im Fall der fachlichen 1:1-Ablösung eines bestehenden Systems. Das gleiche gilt im Fall von Anpassungsprojekten, bei denen eine bestehende (Standard-)Software anzupassen ist, oder im Fall von Integrationsprojekten, bei denen eine Vielzahl von bestehenden Services zu einer neuen Anwendung komponiert wird.

  Das Entsprechende gilt für die anderen Domänen.

- Benötigte Nachhaltigkeit der Architektur:

  Es ist damit zu rechnen, dass ein niedriger Skill-Level eher eine strenge Kapselung von extrem komplexen Sachverhalten verlangt, als ein extrem hoher Skill-Level bei den beteiligten Projektmitarbeitern. D.h. hier wird eine Entwicklungsmethodik verlangt, die sicherstellt, dass vorab z.B. über technische Prototypen etc. abgesichert wird, dass eine tragfähige Architektur mit den entsprechenden Eigenschaften entsteht. Das gleiche gilt, wenn hohe Anforderungen an Performance usw. gestellt werden.

  Auch hier ist es sinnvoll, eine explizite Designphase einzuplanen, bei der die Architektur des künftigen Systems entsprechend den Anforderungen entwickelt und geprüft wird. Dabei sind die übergeordneten Design-Prinzipien festzulegen, um deren Durchgängigkeit über den gesamten Folgeprozess nachhalten zu können. Gleichzeitig ist es in Projekten mit hohen Anforderungen in diesen Bereich sinnvoll bereits erprobte Architekturmuster wiederzuverwenden. Diese können rein technisch geprägt sein oder für einen gesamten fachlichen Typ von Anwendungen gelten.

- Benötigte Nachhaltigkeit der Technologie:

  Wenn sicherzustellen ist, dass die in dem System zur Anwendung kommende Technologie eine sehr lange Lebenszeit hat, während derer wenige fachliche Änderungen auftreten werden, dann ist die Methodik und die zu verwendenden Werkzeuge so zu wählen, dass sichergestellt ist, dass diese auch über die gesamte Zeit stabil bleibt.

  Geringe fachliche Änderungen sollten keine erheblichen Änderungen z.B. an der Entwicklungsumgebung oder dem entstehenden System selbst notwendig machen.

ISBN 978-3-88579-286-4

Dabei ist auch die unterliegende Basis zu berücksichtigen. Je nach Kontext kann diese sich unabhängig von der Anwendung weiterentwickeln (z.B. weil das Betriebssystem aktualisiert wird). Dann sollte eine entsprechende Anwendung möglichst von dieser Basis unabhängig sein. Diese Anforderung steht allerdings häufig in Widerspruch zu Anforderungen speziell an die Benutzerschnittstelle, bei der die gewünschte Komplexität und Aktualität nur mit halbwegs aktuellen Entwicklungsumgebungen erreicht werden kann.

- Art und Umfang der notwendigen Dokumentation:

  Das Sourcing in einem Projekt beeinflusst maßgeblich, wieviel Dokumentation notwendig ist. In einem reinen Onsite-Projekt, wo alle Mitarbeiter wenigstens grundsätzlich ein gleichartiges kulturelles und sprachliches Verständnis haben, sind die Anforderungen an Dokumentation geringer, als in einem Projekt mit deutlichem Offshore-Anteil bei dem auch noch versucht wird, die Mitarbeiter phasen-orientiert einzusetzen (s. auch [SEH10]). Gleichzeitig ist der benötigte Dokumentations-Umfang z.B. auch abhängig von den Vorstellungen, wer die spätere Wartung übernehmen wird (Skill-Level, notwendiger Know-How-Transfer etc.). Das gleiche gilt für die Dokumentation von Entscheidungen. Abhängig von dem System-Kontext genügt es, nur die Ergebnisse von Entscheidungen festzuhalten, in anderen Fällen sind dagegen auch die Gründe für getroffene Entscheidungen und die verworfenen Alternativen zu protokollieren.

- Zulässige Komplexität der Entwicklungsmethodik, maximal geeignete Teamgröße und Teamverteilung:

  Es kann in einem langlaufenden Projekt wirtschaftlich sinnvoll sein, eine komplexere Entwicklungsmethodik mit entsprechenden Werkzeugen zu verwenden, wenn diese den reinen Entwicklungsaufwand reduziert, weil der Aufwand für das Erlernen dann weniger ins Gewicht fällt. Analog benötigt ein Team mit 100 Mitarbeitern eine andere Methodik als eines mit zwei Kollegen, die an einem Tisch sitzen.

Zusammenfassend lässt sich hier feststellen, dass die gewählten Eigenschaften zwei Extrem-Typen von Software-Entwicklung charakterisieren:

- Sehr stark industrialisierte Softwareentwicklung, bei der Wiederholbarkeit, Optimierung der Einzelschritte und der Gesamtkette sowie Nachvollziehbarkeit im Vordergrund stehen und

- einen eher handwerklichen Ansatz, der Effizienz, Wiederholbarkeit und Nachvollziehbarkeit zu Gunsten von Flexibilität hinten anstellt.

Es ist aber relativ klar, dass es durchaus Umfelder gibt, bei denen andere oder nur Teile der oben genannten Eigenschaften von Methodenbausteinen relevant sind. Ein (fiktives) Beispiel dafür wäre ein Umfeld, bei dem sich die Methodenbausteine nur dadurch unterscheiden, in welcher fachlichen Domäne sie eingesetzt werden (getrennte Methodik für Automotive-Anwendungen von der im Bereich Aviation).

Schritt 3: Festlegung von Systemtypen

In diesem Schritt bilden wir Systemtypen abhängig davon, welche Art von Methodenbausteinen sie benötigen (siehe Abb. 6). Eine derartige Einteilung muss nicht für jeden Kontext geeignet sein. Es ist sicher denkbar, andere Gruppierungen zu bilden. Allerdings ist eine solche Einteilung in Systemtypen nur dann hilfreich, wenn sie nicht eine zu große Gruppe von Systemen auszeichnet, weil es sonst relativ schwierig wird, die zu Grunde liegenden Konzepte an die beteiligten Mitarbeiter zu vermitteln und eine zunehmende Beliebigkeit zu verhindern.

Wir identifizieren zwei Extrem-Ausprägungen:

- Der Hightech-Katamaran:

  Diese Projekte zeichnen sich dadurch aus, dass sie in hohem Maße dynamisch mit dem Kunden die angestrebte Fachlichkeit erarbeiten bzw. im weiteren Lebenszyklus immer wieder ändern. Dabei ist die Schnelligkeit der Lösungsentwicklung und der innovative Charakter der Lösung von entscheidender Bedeutung, eine lange Wartungsphase ist nicht im Fokus.

  Umfangreiche Dokumentation ist wenig hilfreich, weil sie sich auf Grund der schnellen Änderungen schnell überlebt und damit einen erheblichen Zusatzaufwand verursacht.

  Typischerweise ist dieser Systemtyp auch mit einer komplexen Benutzeroberfläche verbunden, so dass auch die für die Umsetzung notwendige Technologie und die dafür notwendige Entwicklungsumgebung möglichst „state of the art“ sein muss. Hier ist es eher nicht notwendig, eine nachhaltige Architektur anzustreben, weil diese sich durch die laufenden fachlichen Änderungen immer wieder anpassen wird. D.h. natürlich nicht, dass grundlegende Architektur-Prinzipien vernachlässigt werden dürfen, weil auch solche Anwendungen häufig hohe Anforderungen an ihre Qualität stellen und in eine komplexere Anwendungslandschaft eingebettet sind, die nicht jede Änderung mitmachen kann und soll.

  Hier ist es möglicherweise sinnvoll Effizienz und damit Schnelligkeit über Effektivität zu stellen. Gegebenenfalls wird damit zeitweise ein Entwicklungspfad verfolgt, der am Ende nicht in Produktion geht, aber geholfen hat, zu entscheiden, was die fachlich gewünschte Lösung ist.

ISBN 978-3-88579-286-4

In diesem Bereich geringe Stückkosten (d.h. in der Regel geringe Lohnkosten) anzustreben ist eher kontraproduktiv, weil die Mitarbeiter wegen des hohen Innovationsgrades und der notwendigen Umsetzungsschnelligkeit eher vor Ort arbeiten müssen. Gleichzeitig kann es sehr sinnvoll sein, in diesem Bereich in deutliche Werkzeug-Unterstützung zu investieren, um die angestrebte Geschwindigkeit zu erreichen.

In der Extremausprägung ist die erwartete Lebenszeit des entwickelten Softwaresystems eher geringer, falls sich im Lauf der Zeit herausstellt, dass die Fachlichkeit an Stabilität gewinnt. Dann kommt u.a. eine Re-Implementierung in Frage, die die Eigenschaften eines Tankers aufweist.

- Der Tanker:

  Hier haben wir häufig die Situation, dass in Ablösung eines Bestandssystems ein neues System zu bauen ist, das eine erhebliche Lebenserwartung aufweist. Dies bedingt, dass die Wartung an Gewicht bekommt und damit verbunden auch Fragen der Nachhaltigkeit. Gleichzeitig ist auf Grund der langen Wartungsphase damit zu rechnen, dass die Teamkonstanz sinkt. Es ist also deutlich mehr Aufwand in eine geeignete Dokumentation zu investieren. Gleichzeitig ist hier darauf zu achten, dass auch Technologie, Methodik und Werkzeuge sowie schlussendlich auch insbesondere die Architektur dem geeignet Rechnung tragen.

  Bei diesem Systemtyp gewinnen Effektivität und Preis-Effizienz deutlich an Gewicht.

In der Regel kommt keiner der beiden Extremtypen wirklich in Reinform vor. Tanker haben gerne Benutzeroberflächen, bei denen technische Aktualität für die Nutzer wichtig ist, auch Katamarane haben Bestandteile, bei denen Konstanz wichtig und erreichbar ist.

Daher führen wir zwei weitere Typen ein, die Mischformen darstellen. Diese dienen dazu die Auswirkungen von Größe zu unterscheiden:

- Der Cruise-Liner

  ist die große Mischform. D.h. er hat als Basis eine stabile Tankerstruktur, die eine erhebliche Konstanz aufweist und ergänzt diese durch Komponenten, die schneller Veränderung unterworfen sind. Bei diesen Systemen ist eine rechtzeitige Prüfung, welche Teile der Anwendung in welche Kategorie fallen, sinnvoll, um zu vermeiden, dass die Optimierungspotentiale, die durch eine getrennte Entwicklungsmethodik gehoben werden können, verschenkt werden. Dabei sind diese Optimierungspotentiale allerdings abzuwägen gegenüber Gewinnen durch Gleichartigkeit.

- Die Fähre

  ist die kleine Schwester des Cruise-Liners. Hier ist es in der Regel nicht sinnvoll, weiter hinsichtlich Methodik zu unterscheiden, weil die zu hebenden Synergie-Effekte nicht ausreichen. Stattdessen ist es zweckmäßig zu Beginn festzulegen, welche Seite das größere Gewicht hat und dann die Methodik entsprechend zu wählen.

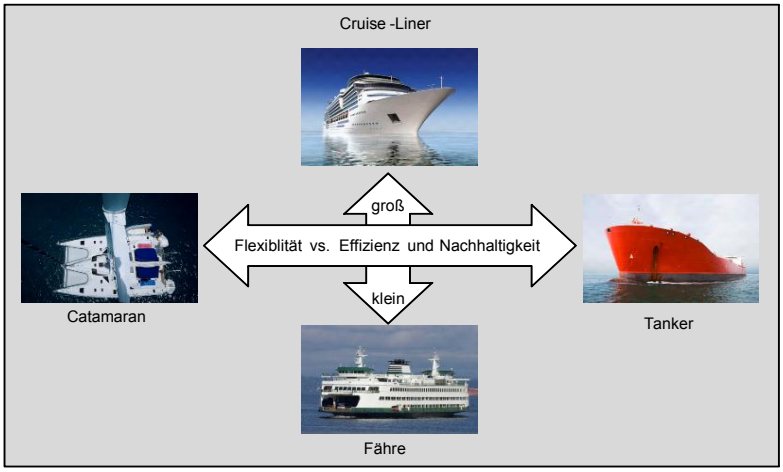

Abb. 6: Systemtypen und ihre Abhängigkeiten im Überblick

Schritt 4: Zuordnung von Methoden und Vorgehensmodellen zu Systemtypen

Aus dem bisher Gesagten lässt sich nun der Zusammenhang zwischen Methoden und Systemeigenschaften herstellen. Abbildung 7 zeigt dies skizzenhaft.

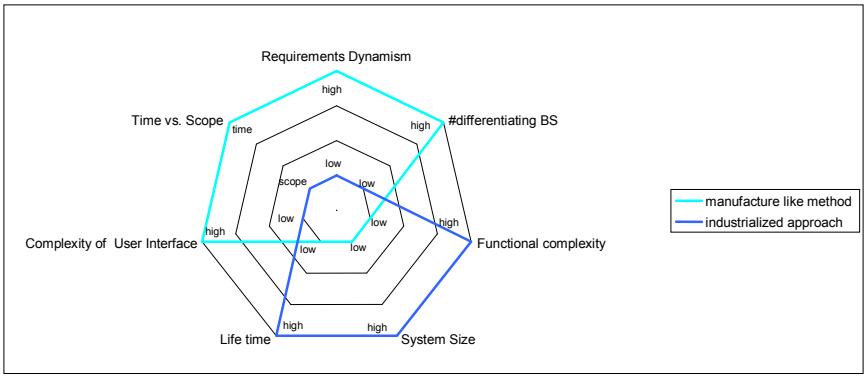

Abb. 7: Zusammenhang zwischen Methoden und Systemeigenschaften

Dabei ist der Zusammenhang zu unserer Differenzierung im Kontext Vorgehensmodelle auffällig. Der handwerkliche Ansatz (manufacture-like method) erscheint sinnvoll in den Bereichen, wo die agile Vorgehensweise ihre Stärken hat, entsprechend passt der industrialisierte Ansatz gut zu iterativen Vorgehensmodellen. Damit wird klar, dass das Vorgehensmodell sich analog ermitteln lässt wie die im Rahmen des Vorgehensmodells zu verwendenden Methodenbausteine.

Auf der Basis des bisher Erarbeiteten kann dann im nächsten Schritt für jeden Systemtyp eine Gesamtmethodik aus den Methodik-Bausteinen pro Domäne und geeignetem Vorgehensmodell zusammengestellt werden.

Es ist dabei davon auszugehen, dass die Gesamtmethodiken nicht überschneidungsfrei sind. So macht es vermutlich Sinn, im Fall des Katamarans und des Tankers die gleichen Methoden und Werkzeuge im Bereich Konfigurationsmanagement zu verwenden, weil sonst Cruise-Liner nicht konsistent einbettbar sind. Gleichzeitig kann analog zu den oben beschriebenen Arbeiten zur Selektion von Vorgehensmodellen eine Anleitung gestaltet werden, wie damit umgegangen wird, wenn ein System in einigen Eigenschaften nicht zu „seinem“ Systemtyp passt. Damit bekommen wir Zusammenhänge wie in der folgenden Abbildung 8 skizziert:

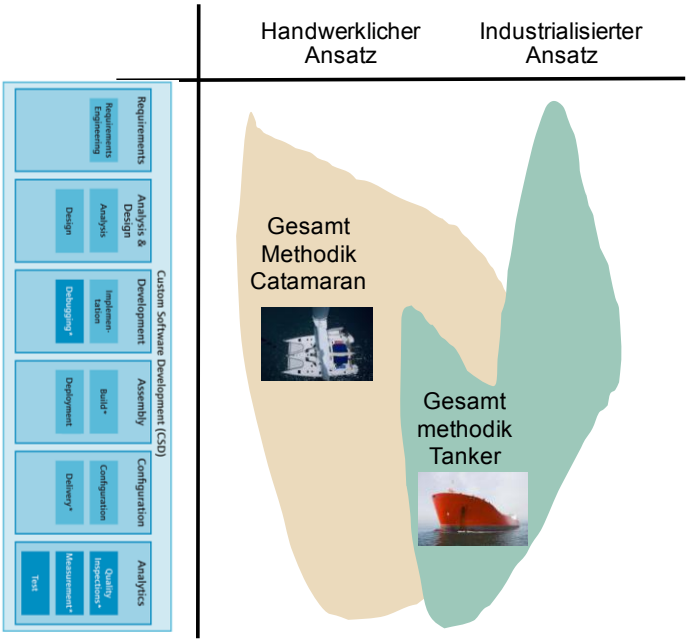

Abb. 8: Zuordnung von Systemtypen zu Vorgehensweisen und Methodenbausteinen

5 Nutzung des Rahmenwerks

Nach der Festlegung des Rahmenwerks, d.h. wenn insbesondere auch die Methoden inkl. Werkzeugen usw. festgelegt sind, kann dies für konkrete Projekte genutzt werden.

Schritt 5: Ermittlung des Systemtyps für ein Projekt

Dafür werden im ersten Schritt die Eigenschaften des konkreten Systems an Hand der strukturellen Dimensionen bestimmt und daraus der Systemtyp ermittelt.

Schritt 6: Auswahl eines situationsbedingten Vorgehensmodells und einer Entwicklungsmethodik

Im letzten Schritt werden dann daraus das passende Vorgehensmodell und zugehörige Methodenbausteine abgeleitet und die Stellen identifiziert, wo durch projektspezifische Maßnahmen sichergestellt werden muss, dass die vorgesehene Methodik auch wirklich eingesetzt werden kann.

6 Zusammenfassung und Ausblick

Wir haben in dem vorliegenden Artikel eine Vorgehensweise vorgeschlagen, um für ein konkretes Systemumfeld eine situationsgerechte Auswahl von Entwicklungsmethodenbausteinen und Vorgehensmodellen zu treffen. Die hier dargestellten Schritte dienen dazu, in Kundenprojekten die optimale, situationsgerechte Vorgehensweise zu definieren.

Zur Zeit werden die beschriebenen Erkenntnisse dazu genutzt, die existierende Entwicklungsmethodik Quasar 2.0 im Sinne eines „Method-as-a-Service“ in wohldefinierte Methodenbausteine zu zerlegen, die dann situationsgerecht zu einer vollständigen projektspezifischen Entwicklungsmethodik zusammengesetzt werden können. Andererseits wird es im Moment nicht als sinnvoll erachtet, für jeden Systemtyp eine komplette Produktionsstraße vorzuhalten, da alleine das bereits einen hohen Aufwand bedeutet, der sich nur dann lohnt, wenn es genügend Projekte in dieser Kategorie gibt und diese dadurch wirklich optimiert werden können. Sollten z.B. alle Projekte in einer Kategorie in Kundenumfeldern stattfinden, in denen die Methodik vorgegeben ist, dann erscheint es nicht sinnvoll, hier Aufwand in die Methodik-Entwicklung zu investieren. Von daher hilft die hier vorgeschlagene Vorgehensweise auch, den Aufwand für die Weiterentwicklung von Software-Entwicklungsmethoden bei Capgemini zielgerichtet zu steuern.

ISBN 978-3-88579-286-4

Das vorgestellte Verfahren stellt darüber hinaus einen ersten Schritt bei der Ausprägung von Architekturmustern dar. Sobald das vorgestellte Konzept nämlich um fachliche Domänen erweitert wird, können ggf. ganze Architektur-Muster für bestimmte fachliche Anwendungsfelder mit vorgegeben werden. Sinnvollerweise wird dies jedoch in einer dritten Dimension ausgedrückt, um unnötige Dopplungen bei der eigentlichen Entwicklungsmethodik und den Vorgehensmodellen zu vermeiden.

Literaturverzeichnis

[Bri96] Brinkkemper, S.: Method engineering: engineering of information systems development methods and tools, Information and Software Technology 38 (4): 275-280, 1996.

[BT03] Boehm, B.; Turner, R.: Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[Cap09] Quasar 2.0: Software Engineering Patterns, Capgemini sd&m, Munich 2009.

[ESS08] Engels, G.; Sauer, St.; Soltenborn, Chr.: Unternehmensweit verstehen – unternehmensweit entwickeln: Von der Modellierungssprache zur Softwareentwicklungsmethode. Informatik-Spektrum, 31(5):451-459. Springer, Berlin/Heidelberg, 2008.

[ES10] Engels, G.; Sauer, St.: A Meta-Method for Defining Software Engineering Methods. In Engels, G.; Lewerentz, C.; Schäfer, W.; Schürr, A.; Westfechtel, B. (eds.): Graph Transformations and Model-Driven Engineering. Springer, LNCS 5765, 411-440, 2010.

[HE10] Heinemann, M.; Engels, G.: Auswahl projektspezifischer Vorgehensstrategien. In O. Linssen, T. Greb, M. Kuhrmann, D. Lange, R. Höhn (eds.): Integration von Vorgehensmodellen und Projektmanagement. Shaker Verlag, 132-142, 2010.

[HR10] Henderson-Sellers, B.; Ralyté, J.: Situational method engineering: state-of-the-art review. J Univers Comput Sci 16(3):424-478, 2010.

[JBR09] Jacobson, I.; Booch, G.; Rumbaugh, J.: The Unified Software Development Process. Addison-Wesley Professional, 1999.

[RB07] Rausch, A.; Broy, M.: Das V-Modell XT – Grundlagen, Erfahrungen und Werkzeuge. dpunkt.verlag, Heidelberg, 2007.

[SEH10] Salger, F.; Engels, G.; Hofmann, A.: Assessments in Global Software Development: A Tailorable Framework for Industrial Projects. In W. Visser, I. Krüger (eds.): Proc. ACM/IEEE 32nd Intern. Conf. on Software Engineering, SE in Practice Track, Cape Town, South Africa (ICSE'10). ACM, NY, USA, vol. 2, 29-38, 2010.

[Sie04] Siedersleben, J.: Moderne Softwarearchitektur: Umsichtig planen, robust bauen mit Quasar. dpunkt.verlag, Heidelberg, 2004.

[OMG08] OMG, Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0, http://www.omg.org/spec/SPEM/2.0/, 2008.

[WK92] Welke, R.J., Kumar, K.: "Method Engineering: a proposal for situation-specific methodology construction". In Cotterman, Senn (eds.): Systems Analysis and Design: A Research Agenda. Wiley, Chichester. 257–268, 1992.