mehr zum thema:
www.qme-Software.de
www.apm4mda.org

AGILE MODELLBASIERTE SOFTWAREENTWICKLUNG IN DER PRAXIS

Wenn die Entwicklungsprozesse bereits nach MDA/MDD-Ansatz (Model Driven Architecture bzw. Development) ausgerichtet sind und eine entsprechende MDA-Tool-Kette existiert, dann ist agiles Projektmanagement (APM) für die modellbasierte Softwareentwicklung ein weiterer folgerichtiger Schritt in Richtung „professionelle MDA-Projekte". In diesem Rahmen sind das Projektmanagement und das Zusammenspiel mit der Softwareentwicklung zu optimieren. Der Artikel zeigt einen praxisnahen Ansatz, um auf der Basis des „Eclipse Process Framework" einen MDA-Entwicklungsprozess für das Projektmanagement bereitzustellen und dabei konsequent agil und damit flexibel zu bleiben. Wir nennen diesen Ansatz APM4MDA.

Agilität – was war das doch gleich?

Agile Ansätze existieren mittlerweile in sehr unterschiedlicher theoretischer und praktischer Ausprägung: Da werden einige „gute alte" *XP*-Techniken (*eXtreme Programming*) in Verbindung mit anderen, eher schwergewichtigen Vorgehensmodellen verwendet oder es gibt Scrum-Projekte, die der reinen Lehre folgen. Bei der Vielfalt der Verfahren und Varianten fällt es schwer, die agilen Gemeinsamkeiten und *Best Practices* zu erkennen und ein optimales Vorgehen zu finden. Denn weder das blinde Mitlaufen und kritiklose Akzeptieren der Agilität, noch eine unreflektierte Abwehrhaltung sind hilfreich. Das Fehlen substanzieller sozialer, organisatorischer und methodischer Bausteine im Softwareentwicklungsprozess macht jeden Ansatz – ob agil oder nicht agil – unglaubwürdig und führt letztlich nur zur „Wurschtelei". Das gilt sowohl für den Engineering-Prozess als auch für Querschnittsaufgaben, wie das Projekt- und Qualitätsmanagement.

Was steckt nun eigentlich hinter dem Begriff „Agilität" und wie grenzt dieser sich von den traditionellen Ansätzen ab? Keine Sorge: An dieser Stelle wollen wir nicht die altbekannten Prinzipien des Agilen Manifestes ausbreiten, z. B. „Business people and developers must work together daily throughout the project" (vgl. [Bec01]), sondern eher einige kurze Anmerkungen machen, die für das Verstehen von APM4MDA hilfreich sind:

- *Kein agiler Standard:* Bei den agilen Ansätzen gibt es kein offizielles Standardverfahren, sehr wohl aber Standardisierungsbemühungen, z. B. mit den „Scrum Alliance Certifications" (vgl. [Scr]).
- *Kombiniertes Vorgehen:* Traditionelle (schwergewichtige) Vorgehensmodelle, z. B. das V-Modell XT (vgl. [VMXT]), und agile (leichtgewichtige) Vorgehensweisen sind kein Widerspruch und lassen sich durchaus kombinieren.
- *Regelwerk:* Auch die agilen Vorgehensweisen kommen mit klaren Vorgaben daher (vgl. [Oes08]) und sollten nicht in einer Beliebigkeit oder Anarchie enden, wo jeder alles darf.
- *Externe und interne Sicht:* Teams mit agilen Vorgehensweisen sind intern basisdemokratischer und selbst organisierter als bei traditionellen Ansätzen. Nach außen muss dies jedoch nicht zwingend „sichtbar" sein.
- *Werte im Vordergrund:* Agile Ansätze betonen den Menschen und die Werte, nach denen gehandelt wird, und unterbreiten durchaus auch konkrete Vorschläge zum Vorgehen, liefern jedoch kein umfassendes Wertesystem im Sinne einer Leitkultur.

Besonders die zuweilen heraufbeschworene Polarisierung, die die traditionellen Vorgehensmodelle (Stichwort „Wasserfall") von den agilen Verfahren trennt, führt zu Glaubenskriegen und ist wenig zielführend, denn es bestehen viele Gemeinsamkeiten und sie lassen sich in Einklang bringen. So bietet das V-Modell XT beispielsweise eine Projektdurchführungsstrategie für die agile Systementwicklung an, wobei es vor einer Bewertung allerdings weiterer praktischer Erfahrungen bedarf.

Agiles Projektmanagement

Im Folgenden wollen wir uns weniger mit den Werten der Agilität befassen als mit den handfesten Praktiken, die sich auf Prinzipien stützen, die schon lange vor den agilen Vorgehensweisen bekannt waren: Dies sind beispielsweise die iterativ-inkrementelle Entwicklung, Timeboxing, die Priorisierung von Anforderungen und die intensive sowie frühe Einbindung des Kunden bzw. der Endanwender. Aus Platzgründen können hier nur einige wenige Techniken Erwähnung finden, die für das Projektmanagement von Bedeutung sind.

Agiles Projektmanagement betrachtet nicht nur Prozesse, sondern auch Produkte, die wir in Anlehnung an den *Rational Unified Process (RUP)* Artefakte nennen. Die Basis sind Vorlagen, die für das gesamte

die autoren

Prof. Dr. Roland Petrasch
(E-Mail: petrasch@tfh-berlin,de)
lehrt und forscht an der TFH Berlin im Bereich modellbasierte Softwareentwicklung sowie Projekt- und Qualitätsmanagement.

Florian Fieber
(E-Mail: florian.fieber@qme-software.de)
ist Berater bei der qme Software in Berlin und beschäftigt sich mit Qualitätsmanagement im Rahmen von MDA-Projekten.

Maciej A. Bednarz
(E-Mail: mab@hacon.de)
ist Softwareentwickler bei der HaCon Ingenieuresellschaft mbH in Hannover und beschäftigt sich mit den Themen JEE/ SOA-Architekturen, Softwaretest und angewandte MDA.

Projekt gelten. Papier wird dabei kaum noch produziert – Artefakte sind bei modellgetriebenen Projekten eigentlich „nur noch" abstrakte Sichten auf ein zentrales Modell-Repository. In konkreter Ausprägung können sich diese in Dokumenten wiederfinden, jedoch sind auch andere Formen, wie Build-Ergebnisse oder z. B. auch elektronische Protokolle, möglich.

Kommen wir zu den Phasen, die ein agiles Projekt durchlaufen sollte. Neben den Phasen sind auch einige Artefakte genannt:

- *Vorbereitungsphase:* Feature-Katalog, Domänenmodelle, Projektplan
- *Startphase:* detaillierte Features, Prototyp, Softwarearchitektur
- *Hauptphase* mit der iterativ-inkrementellen Entwicklung: Integrations-Builds, Releases
- *Abschlussphase:* Abnahmetestprotokoll

Entscheidende Artefakte zu Beginn eines Projekts sind unter anderem die Use-Cases und die Feature-Liste, auf deren Basis die Aufwandsschätzung (*Widget-* oder *Use-Case-Point*-Technik) möglich ist. Häufig zum Einsatz kommende Methoden, die von historischen Daten ausgehen (z. B. die Analogiemethode), sind bei den ersten MDA-Projekten kaum verwendbar.

Wir haben die Erfahrung gemacht, dass MDA-Projekte ohne die Aufwände für die MDA-Infrastruktur, mindestens 20 bis 30 % weniger Kosten verursachen als ähnliche Projekte ohne MDA-Ansatz. Dies deckt sich auch mit anderen Erfahrungen (vgl. z. B. [Fri06]). Insofern kann durchaus zunächst „klassisch" geschätzt werden, wobei allerdings das MDA-Infrastrukturprojekt zu beachten ist, das eine separate Investition darstellt – dazu aber später mehr.

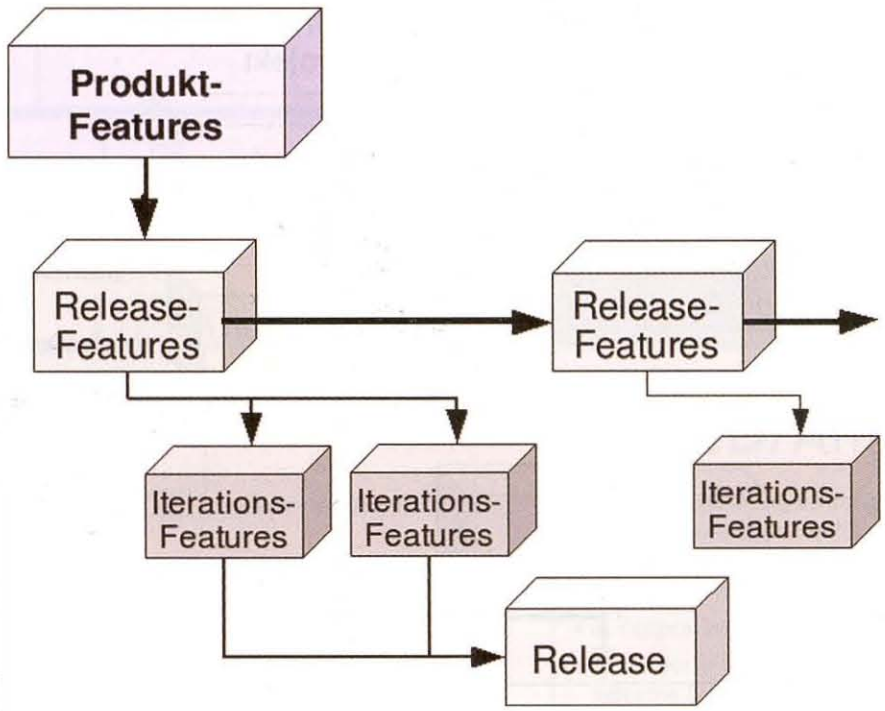

Abb. 1: Produkt-, Release- und Iterations-Features.

Wichtig für die erste Projektplanung sind die Produkt-Features, die in Release-Features unterteilt werden. Ausgehend von der Release-Planung lassen sich dann Iterationen mit Iterations-Features planen. Die Iterationen (mit einer Dauer von ca. einem Monat) planen die Teams eigenständig, wobei sich hier das *Timeboxing*-Verfahren bewährt hat: Die Iteration wird dabei zeitlicher „Fixstern", d. h. als nicht veränderbares Zeitfenster definiert (siehe Abb. 1). Bei der Iterationsplanung werden bestimmte Ergebnisse (in der Regel Artefakte) vorgesehen. Stimmen dann die tatsächlich erstellten Artefakte nicht mit den geplanten überein, so erfolgt eine Übertragung in die nächste Iteration (aber keine zeitliche Verschiebung der Iteration selbst). Die Diskrepanzen ermöglichen Rückschlüsse für die nächste Planung im Sinne einer Prozessverbesserung.

Meilensteine kennzeichnen einen irreversiblen Status eines Projekts, d. h. ein bestimmter Stand – beispielsweise eine Menge an umgesetzten Release-Features – ist vorhanden. Meilensteine sind also nicht zwingend zeitlich fixiert, wobei es natürlich das Ziel sein muss, die Meilenstein-Termine zu halten. Sie werden unter anderem jeweils am Ende eines Releases geplant.

Eine hohe Prozessqualität ist dann erreicht, wenn innerhalb aller Timeboxen die jeweiligen Iterations-Features umgesetzt sind und am Ende alle Release-Features ▸

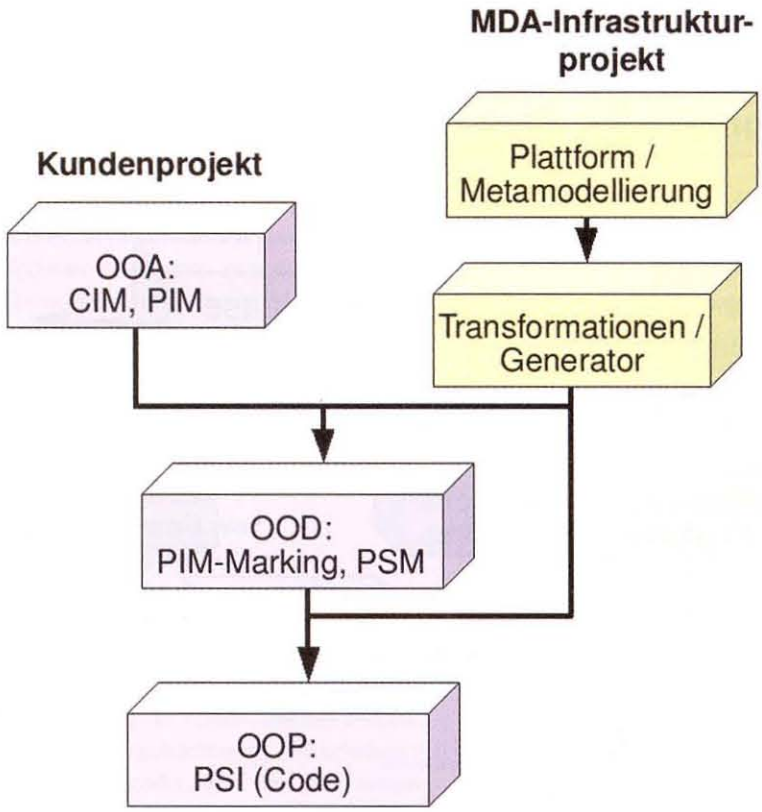

Abb. 2: Y-Modell für MDA-Projekte mit Kunden- und MDA-Infrastrukturprojekt (in Anlehnung an [Pet06]).

pünktlich vorliegen, sodass sich der geplante Meilenstein nicht verschiebt.

Grenzen der Agilität

Dies alles darf aber nicht über eines hinweg täuschen: Zuweilen gibt es handfeste Verklemmungen zwischen Prozess- und Vorgehensmodellen einerseits und den agilen Ansätzen andererseits. Das drückt sich in kaum vereinbaren Prozessen, Rollen oder Artefakten aus. Im Folgenden wird ein solcher Konflikt am Beispiel des Prozessreferenzmodells *Automotive SPICE* und *SCRUM* (vgl. [SPI05] und [Pic08]) gezeigt.

In der Literatur und auch in realen Scrum-Projekten ist das Team für den Aufgabenbereich Qualitätsmanagement zuständig (vgl. [Pic08], S. 24, 122). Dies lässt sich jedoch nur schwer mit der Anforderung des Prozessmodells Automotive SPICE vereinbaren: Hier wird eine Trennung beim *Supporting Process SUP.1* (Qualitätssicherung) explizit gefordert: „Quality assurance team members are not directly responsible to the project organisation – they work independently from it" (vgl. [SPI05], S. 53).

Ein weiterer Punkt ist die Terminologie: XP beispielsweise lässt sich zwar theoretisch auf MDA abbilden, aber die Code-Zentrierung ist dem XP „auf die Stirn" geschrieben – eine extreme Modellierung will daraus einfach nicht werden, zumal essenzielle Bestandteile der MDA fehlen, z. B. DSL-Entwicklung, Metamodellierung oder Referenzimplementierung. Weder bei den Primär- noch bei den Folgetechniken des XP findet sich so etwas (vgl. [Bec04]).

Weiterhin müssen die agilen Techniken auch im Team vermittelt werden, da sonst Unverständnis entstehen kann: Die zeitlich fixierte Iteration scheint zu dem agilen Wert „Responding to change over following a plan" (vgl. [Bec01]) im Widerspruch zu stehen, was jedoch nicht stimmt, denn wichtig sind die Iterationsergebnisse, die sehr wohl flexibel vom Plan abweichen können, wenn es beispielsweise Änderungen bei den Anforderungen gab. Das Kommunizieren und Verstehen solcher Mechanismen ist essenziell, kommt aber eben nicht von selbst bzw. „über Nacht".

Modellbasierte Softwareentwicklung

Die grundlegenden Ideen der modellgetriebenen Entwicklung sind im MDA-Guide der Object Management Grouip (vgl. [OMG03]) einführend erklärt, aber ein einheitliches MDA/MDD-Vorgehensmodell, bei dem in standardisierter Form Strategien und Prozesse festgelegt sind, sucht man zur Zeit noch vergeblich. Die Anwendungsmöglichkeiten sind derart breit (z. B. „Model-to-Model", „Model Merging", „MDA Reverse Engineering"), dass derartiges wohl auch erst in den nächsten Jahren sukzessive entstehen wird. Ansätze mit ähnlichem Namen, z. B. *Model Driven Software Development (MDSD)*, stellen letztlich nur Spezialfälle der MDA dar.

Es gibt allerdings einige sich mehr oder weniger zwingend ergebende Aktivitäten, Artefakte und Rollen, etwa Softwarearchitekt oder (Meta-)Modellierer. Unterstellt man einem Prozess ein konsequentes Forward-Engineering mit dem Ziel der Softwaregenerierung auf der Basis von Modellen, so bietet sich zunächst als Veranschaulichung das *Y-Modell* an (siehe Abb. 2). Dabei ist zwischen zwei grundlegenden Projekten zu unterschieden:

- dem Kundenprojekt und
- dem MDA-Infrastrukturprojekt (auch Generatorprojekt genannt).

Das Kundenprojekt umfasst die typischen MDA-Aktivitäten: die plattformunabhängige Modellierung der fachlichen Welt der Anwender (CIM, PIM), die Transformation zum plattformabhängigen Zielarchitekturmodell (PSM) und schließlich die Codegenerierung (PSI). Das MDA-Infrastrukturprojekt besteht aus der Entwicklung von domänenspezifischen Sprachen (DSLs), UML-Metamodellderivaten oder UML-Profilen sowie der Transformationsspezifikation und -implementierung, sodass dann alles schön „verpackt" als Generator zur Verfügung steht.

MDA-Aktivitätsbeispiel

Um eine Iteration im Kontext des Y-Modells näher zu erläutern, sei ein typisches (aber stark vereinfachtes) Vorgehen bei MDA-Projekten als Aktivitätsdiagramm mit Hilfe des *Eclipse Process Frameworks (EPF)* dargestellt (siehe Abb. 3): Dabei geht es um die Referenzimplementierung, d. h. die manuelle Entwicklung eines kleinen Beispielprojekts, um die Zielarchitektur genau kennenlernen zu können. Im Rahmen dieser Aktivität sind unter anderem die Zielarchitektur zu definieren, die zu verwendenden Frameworks zu evaluieren und die Anwendung manuell zu implementieren. Anschließend kann auf dieser

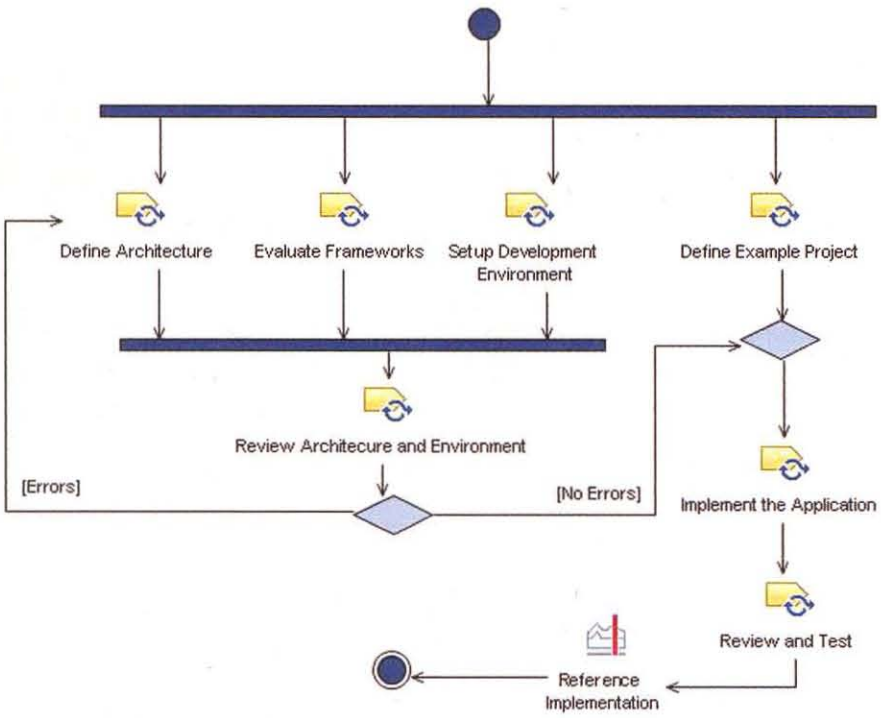

Abb. 3: MDA-Aktivität Referenzimplementierung als Iteration.

Basis der Generator entwickelt werden, der in Zukunft eine manuelle Implementierung weitestgehend unnötig macht. Nicht aufgeführt ist bei diesem vereinfachten Ablauf beispielsweise das Testen, aber es wurde deutlich, wie eine Iteration definiert ist. Sie wird als Timebox geplant. Es ist durchaus möglich, die Referenzimplementierung in mehreren Iterationen zu erstellen. Am Ende steht ein Meilenstein, der die Fertigstellung der Implementierung symbolisiert.

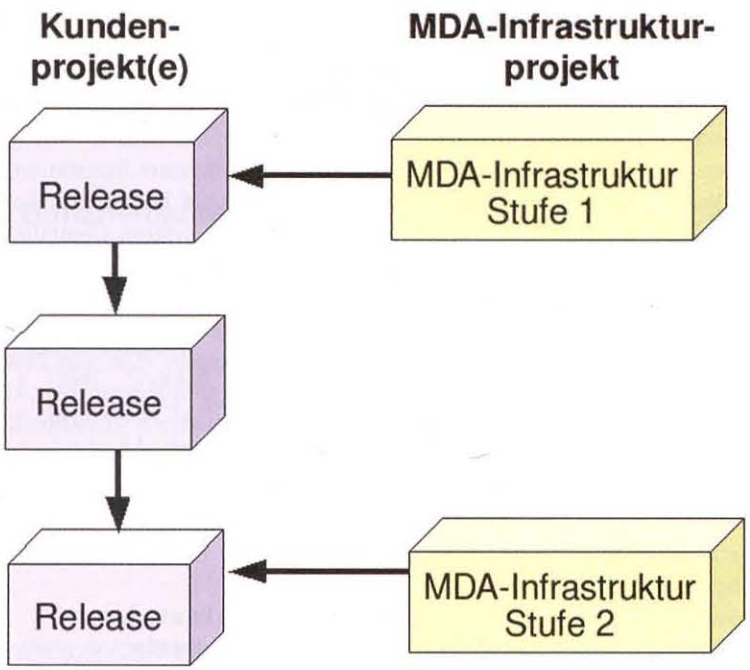

Abb. 4: Parallelentwicklung von Kundenprojekten und MDA-Infrastruktur.

Fortsetzung auf Seite 42 ▶

MDA: parallele Entwicklung

Aber nun zurück zum „großen Ganzen": Das reine Y-Modell ist – so wie das Wasserfallmodell auch – wenig praxisnah, da es weder die iterativ-inkrementelle Entwicklung darstellt, noch den beiden Teilprojekten die notwendige Autonomie gibt: Der Aufbau und die Weiterentwicklung einer MDA-Werkzeugkette und einer passenden Infrastruktur verläuft zwar koordiniert, aber asynchron zu Kundenprojekten bzw. der Produktentwicklung. Letzteres ist ja in der Regel das Ziel einer industriellen Softwarefertigung auf der Basis von modellbasierten Verfahren. Abbildung 4 zeigt die nebenläufige Weiterentwicklung der MDA-Infrastruktur, sodass bei der Release-Planung der parallel dazu verlaufenden Kundenprojekte flexibel entschieden werden kann, in welcher Ausbaustufe die MDA-Tools zum Einsatz kommen – so werden MDA-Projekte agil.

Diese Stufen der MDA-Infrastruktur lassen sich je nach Domäne, Technologie und individuellen Anforderungen gestalten. So können sie zum Beispiel einen zunehmend höheren Codegenerierungsanteil bedeuten:

- Erste MDA-Gehversuche (Stufe 1) bestehen in der Regel darin, nur die Domänenklassengerüste und die Persistenzschicht automatisiert aus den Modellen (PIM) zu erzeugen.
- Die Weiterentwicklung (Stufe 2) umfasst dann Teile der Geschäftslogikschicht (z. B. Zugriffsfassaden) und einige Aspekte der Benutzungsoberfläche, z. B. den Web-Flow.
- Stufe 3 könnte dann auch die kompletten Geschäftsregeln, Algorithmen und die Benutzungsoberfläche umfassen, beispielsweise in Form von *HCI*-Patterns (*Human Computer Interaction*, vgl. [Pet07]).

Eine Stufe ist aus Sicht des Projektmanagements dabei zunächst nichts anderes als ein Release. Komplexer wird es dann, wenn es unterschiedliche Zielarchitekturen und umfangreiche Projekte bzw. Poduktlinien gibt, was natürlich dann auch Auswirkungen auf die MDA-Infrastruktur hat. Wir wollen uns im Folgenden aber auf kleinere (weniger als 8 Mitarbeiter) bis mittlere Projekte (8 bis 20 Mitarbeiter) beschränken. Folgende Teams und Rollen sind dabei vorhanden:

- *Entwicklungsteam für das Kundenprojekt:* Hier sind Rollen wie Systemanalytiker, Softwareentwickler und Tester vorhanden. Die Gewichtung der beteiligten Rollen und die Ausprägung der Aktivitäten des Entwicklerteams verändern sich über den Zyklus eines MDA-Projekts. Während der initialen Phase kann die Erstellung eines *Proof of Concepts* im Vordergrund stehen. Diese mündet bei ausreichender Abdeckung der initialen Anforderungen in der Ableitung entsprechender wieder verwendbarer Softwareartefakte, die auf einer höheren Ebene als der Kodierung logisch in Modellen dargestellt werden.
- *Technikteam für die MDA-Infrastruktur:* Zu diesem Team gehören Softwarearchitekten, (Meta-)Modellierer und (MDA-)Tool-Experten. Das Modellierungsteam übernimmt die technischen Artefakte der initialen Kodierungsebene und definiert daraus ein logisches Modell, das als Steuerung für den parallel zu erstellenden Generator dient. Die Aktivitäten des MDA-Teams sind sinnvollerweise phasenverschoben zu den Aktivitäten des Entwicklungsteams, jedoch aufgrund der inkrementellen Vorgehensweise nicht von diesen entkoppelt.
- *Projektleitung:* Dies kann zwar im Einzelfall eine einzelne Person (Projektleiter) sein, sollte bei MDA-Projekten allerdings noch die Kundenprojekt- und technische Team-Leitung mit einschließen.

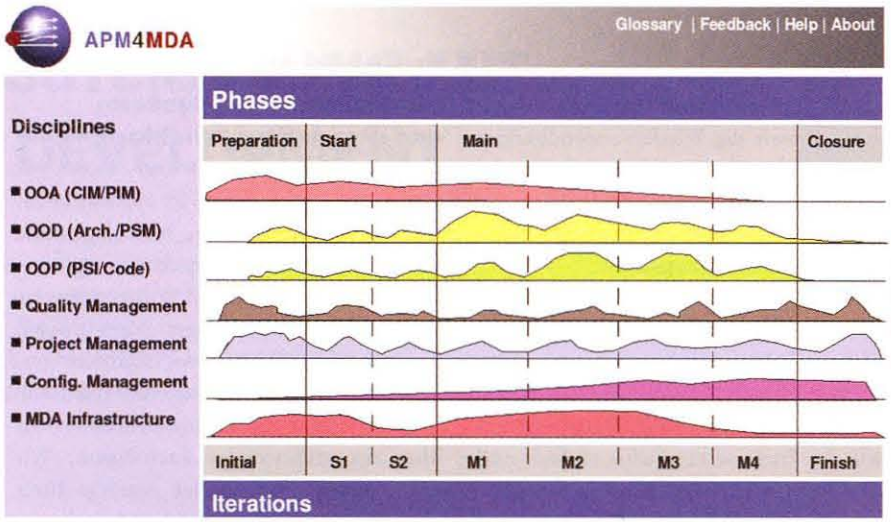

Abb. 5: Methoden und Lebenszyklus von APM4MDA.

Nach der Betrachtung einiger MDA-Prozesse und -Rollen kehren wir zunächst noch einmal kurz zum Konzept der Vorgehensmodelle zurück.

UMA und SPEM

Ein Vorgehensmodell definiert Methoden der Softwareentwicklung, d. h. Beziehungen zwischen den Elementen „Rolle", „Aufgabe" und „Produkte" (Projektartefakte). Durch die logische und zeitliche Anordnung der Methoden entsteht ein Lebenszyklus. Ein Vorgehensmodell besteht somit zum einen aus statischen, vom Lebenszyklus unabhängigen Inhalten (Methoden) und zum anderen aus deren konkreter, dynamischer Anwendung in Prozessen (Lebenszyklus). Diesem Paradigma folgen die bekannten Vorgehensmodelle der Softwareentwicklung, z. B. das V-Modell XT, der RUP, aber auch agile Ansätze, wie XP. Obwohl die genannten Vorgehensmodelle auf diesem Prinzip basieren, sind sie nicht kompatibel zueinander, d. h. sie verwenden unterschiedliche Metamodelle. Zwar lassen sich die Elemente und Konzepte oft aufeinander abbilden, dies hat jedoch auch seine Grenzen, unter anderem bei der Kombination oder Integration der Inhalte.

Die *Unified Method Architecture (UMA)* (vgl. [Shu07]) versucht sich an einer Lösung dieses Problems: Ein Metamodell zur Spezifikation von Methoden und Prozessen ermöglicht die Definition verschiedener Vorgehensmodelle bzw. deren Inhalte. Auch eine Integration unterschiedlicher Ansätze ist denkbar. Die UMA ist eine Weiterentwicklung der Version 1.1 des *Software Process Engineering Metamodel (SPEM)* der

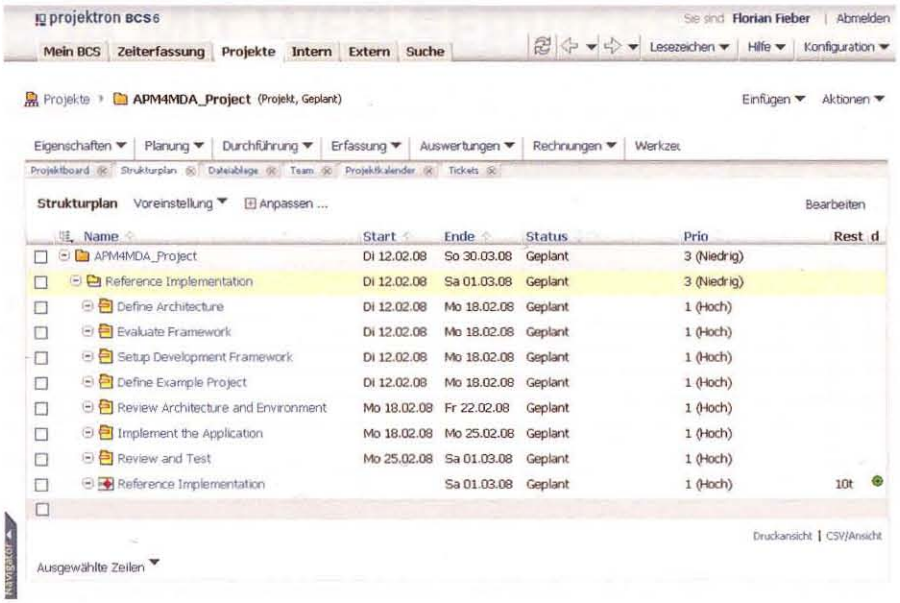

Abb. 6: Mikro-Controlling mit Hilfe des Ampelstatus in Projektron BCS.

OMG. Große Teile der UMA sind ihrerseits in die Version 2.0 der SPEM eingeflossen (vgl. [OMG07]).

EPF: Basis für APM4MDA

Das *Eclipse Process Framework (EPF)* der Eclipse Foundation basiert auf der UMA (und zukünftig auf SPEM 2.0) zur Definition von Methoden und bietet somit ein generisches Prozessmanagementwerkzeug zur Definition beliebiger Vorgehensmodelle. EPF ist eine freie Version des kommerziellen Werkzeugs „Rational Method Composer" von IBM. Das aktuelle Release des EPF enthält den „EPF Composer" und „OpenUP" (Open Unified Process, vgl. [Ecl-a] und [Ecl-b]). Während der EPF Composer ein Prozessdefinitionswerkzeug darstellt, handelt es sich beim OpenUP um einen minimalen, ausgeprägten und erweiterbaren iterativen Softwareentwicklungsprozess im Sinne eines Vorgehensmodells, der die Basis für eigene Erweiterungen und Anpassungen darstellt.

Das Grundprinzip des EPF ist die Trennung von wieder verwendbarem Inhalt in Form von Methoden zum einen und deren Anwendung in Prozessen zum anderen. Die Methoden werden im *Method Content* verwaltet, in *Processes* lassen sich unterschiedliche Lebenszyklen definieren. Abbildung 5 zeigt das MDA-Vorgehen für Kundenprojekte: Die Iterationen finden innerhalb der schon erwähnten Phasen des APM statt: Vorbereitungs-, Start-, Haupt- und Abschlussphase. Der Methodenteil, der sich in den Disziplinen (Disciplines) wiederfindet, ist überwiegend aus Abbildung 2 bekannt.

Eine Methode im *Method Content* stellt eine Anleitung für einen bestimmten Bereich oder eine bestimmte Disziplin der Softwareentwicklung dar, z. B. die Erstellung des plattformunabhängigen Modells (PIM). Sie wird durch die Elemente „Rolle" (z. B. *Domain Analyst, Modeler*), „Aufgabe" (z. B. *Define Product Features, Model Use Cases*) und „Arbeitsergebnis" (z. B. *Feature-Liste, Use-Cases*) sowie deren Beziehungen untereinander beschrieben und lässt sich durch zusätzliche Dokumentation (z. B. *Richtlinien, Checklisten*) ergänzen. Die Aktivitäten der Prozessdimension sind in einem hierarchischen Projektstrukturplan (*Work Breakdown Structure*) abgebildet. Sie enthalten Referenzen auf die Methodeninhalte.

Das EPF bietet somit als generisches Framework zur Erstellung von Prozessen der Software- bzw. Systementwicklung eine gute Grundlage für die Definition von APM4MDA. Die Inhalte und Prozesse lassen sich in Form eines web-basierten Handbuches exportieren, das beispielsweise im Intranet jedem Prozess- und Projektbeteiligten zur Verfügung steht. Aufgabenbeschreibungen, Rollendefinitionen und Dokumentvorlagen sind somit an zentraler Stelle für jeden zugänglich; das Werkzeug wird dadurch zu einer wichtigen Grundlage eines effektiven Software-Qualitätsmanagements.

Literatur & Links

[BecO1] K. Beck et al., Manifesto for Agile Software Development, 2001, siehe: www.agilemanifesto.org

[BecO4] K. Beck, Cynthia Andres: Extreme Programming Explained. Embrace Change (2. Aufl.), Addison-Wesley 2004

[Ecl-a] The Eclipse Foundation, Eclipse Process Framework Project, siehe: www.eclipse.org/epf

[Ecl-b] The Eclipse Foundation, Open Unified Process, siehe: epf.eclipse.org/wikis/openup/

[FriO6] P. Friese, Erfahrungsbericht: Einsatz des Open Source MDA Generators Andro MDA im Projekt Kombiverkehr KMS bei Lufthansa Systems, in: F. Fieber, R. Petrasch, W. Neuhaus, Tagung der SIG MDSE, Logos Verlag 2006

[OesO8] B. Oestereich, C. Weiss, APM – Agiles Projektmanagement – Erfolgreiches Timeboxing für IT-Projekte, dpunkt.verlag, 2008

[OMGO3] Object Management Group (OMG), MDA Guide, Version 1.0.1, 2003, siehe: www.omg.org/mda

[OMGO7] Object Management Group (OMG), Software & Systems Process Engineering Metamodel Specification, v2.0 (Beta 2), ptc/2007-08-07, 2007, siehe: www.omg.org/docs/ptc/07-08-07.pdf

[PetO6] R. Petrasch, O. Meimberg, Model Driven Architecture, dpunkt.verlag, 2006

[PetO7] R. Petrasch, Model Based User Interface Design: Model Driven Architecture und HCI Patterns, in: GI Softwaretechnik-Trends, Mitteilungen der Gesellschaft für Informatik, Band 27, Heft 3, 9/07

[PicO8] R. Pichler, Scrum – Agiles Projektmanagement erfolgreich einsetzen, dpunkt.verlag, 2008

[Pro] Projektron BCS, siehe: www.projektron.de

[Scr] Scrum Alliance, Inc., www.scrumalliance.org

[SPIO5] SPICE User Group, The Procurement Forum: Automotive SPICE. Process Reference Model, Process Assessment Model, 2005

[ShuO7] A. K. Shuja, J. Krebs: IBM Rational Unified Process – Reference and Certification Guide: Solution Designer, IBM PressPub, 2007

[VMXT] Bundesministerium des Inneren (BMI), V-Modell XT, siehe: www.v-modell-xt.de

Übergang zum Projektmanagement

MDA erfordert aus unserer Sicht ein agiles Projektmanagement. Die Entwicklung von

Generatoren geht mit fortlaufenden Tests und notwendigen Anpassungen am Metamodell bzw. den Transformationen einher und wird von zahlreichen Rückkopplungen der Iterationen untereinander begleitet. Auf der Basis des mit dem EPF-Composer definierten APM4MDA-Vorgehens sind für konkrete Projekte nun quasi Prozessmodellinstanzen zu bilden, mit denen in einem Projektmanagement-System weitergearbeitet werden kann, wobei sicherzustellen ist, dass die Agilität erhalten bleibt. Wir verwenden hierfür die Software „Projektron BCS" (vgl. [Pro]): Nach dem Import des EPF-basierten MDA-Vorgehensmodells können die notwendigen Planungsaktivitäten erfolgen, z. B. Meilenstein-, Release- und Iterationsplanung. Wichtig ist dabei ein fein-granulares Rechtesystem, das den Teams ein selbst organisiertes lokales Projektmanagement ermöglicht. Jedem Teammitglied ist damit die notwendige basisdemokratische Freiheit überlassen, wobei die Projektleitung einen Überblick über alle Teams hat und somit die Auskunftsfähigkeit und Steuerbarkeit sichergestellt sind – gerade bei großen IT-Projekten, bei denen APM zum Einsatz kommt, ist dies eine essenziell wichtige Funktionalität.

Abschließend sei noch darauf hingewiesen, dass es auch oder gerade bei agilen Projekten wichtig ist, jederzeit zu wissen, wo das Projekt steht: Wöchentliches Mikro-Controlling ermöglicht den Entwicklern eine Rückmeldung, unter anderem in Form von Restaufwandsschätzungen. Der Ampelstatus signalisiert dann in sehr anschaulicher Form, ob und wo die Projektleitung eingreifen muss, damit alles wieder im „grünen Bereich" ist. Abbildung 6 zeigt den Ampelstatus anhand einiger Aufgaben.

Fazit

Modellbasierte Softwareentwicklung kann (oder besser: muss) durch agiles Projektmanagement professionalisiert und auch für größere Projekte erschlossen werden, wenn ein MDA-spezifisches Vorgehen und geeignetes – d. h. erfahrendes – Personal zur Verfügung stehen. Zwar sind Werkzeuge nicht alles, aber auch bei MDA-Projekten gilt: Ohne Tools ist alles nichts. Die Verbindung zwischen EPF-basierter Prozessdefinition und umfassendem Projektmanagement mit Projektron BCS hat sich als praktikabel erwiesen – die Prozesse lassen sich flexibel erstellen und im Projekt leben. Wie so oft ist es nicht mit einem einzelnen Werkzeug getan, sodass es auf die passende Kombination und die nahtlose Integration ankommt.

Den APM4MDA-Ansatz konnten wir hier zwar nur oberflächlich skizzieren und einige wichtige Aspekte – etwa die Themen Aufwandsschätzung, Release-, Iterations- und Meilensteinplanung, Aufgabe und Retrospektive sowie Abnahmen – konnten gar nicht behandelt werden, aber ein erster hilfreicher Eindruck konnte sicherlich vermittelt werden. ■

www.fruchtbringende-gesellschaft.de/sprachtag.html

GEDANKEN ZUR SPRACHE

Georg Heeg
www.heeg.de

Liebe Leserin und lieber Leser,

in vielen Gesprächen ist mir – als Teilnehmer des durch den ersten Köthener Sprachtags Sensibilisiertem – aufgefallen, dass wir, die wir nicht Englisch als Muttersprache haben, uns schwer tun, die Bilder zu verstehen, die hinter den für uns neuen Fremdwörtern stehen (manchmal auch hinter den alten); ich wurde nämlich neulich gefragt, für was denn Scrum eine Abkürzung sei.

Eine erste Reaktion hätte sein können, den Frager als ungebildet abzustempeln. Das halte ich jedoch für falsch. Ungebildet – vielleicht auch überheblich oder faul – sind doch eher diejenigen, die sich nicht der Mühe unterziehen, passende Wörter in der eigenen Muttersprache zu suchen.

Ich will daher einfach an dieser Stelle einmal den Versuch wagen, einen Teil der englischen Scrum-Begriffe ins Deutsch zu übersetzen:

- *Scrum* heißt wörtlich Gedränge, doch *Drang* scheint mir die bessere Übersetzung zu sein, denn sie ist kürzer und kraftvoller.
- *Backlog* ist eigentlich eine Arbeitsrückstandsliste. Mir ist das jedoch zu abwertend und ich schlage daher *Aufgabenliste* vor.
- Damit wird dann aus *Backlog Item* eine *Aufgabe*.
- *Prioritization* ist naheliegenderweise im Deutschen die *Wichtigkeit*.
- *Estimation* lässt sich problemlos mit *Schätzung* übersetzen.
- *Planning* – das versteht sich von selbst – ist *Planung*.
- *Testability* mit *Prüfbarkeit* zu übersetzen, liegt auf der Hand.
- Eine *Impediment List* würde ich Hindernisliste nennen.
- Für *Sprint* schließlich erscheint mir *Spurt* treffend zu sein.

Habe ich wichtige Wörter vergessen?

Natürlich werden auch wir in internationalen Projekten weiterhin die englischen Begriffe verwenden, aber wenn wir ein Projekt mit Drang betreiben, stets unsere Aufgabenliste nach Wichtigkeit im Spurt bearbeiten, den Aufwand schätzen und die Ergebnisse prüfen, kann der Erfolg nicht ausbleiben.

Herzliche Grüße

Ihr Georg Heeg