Andreas Lux

(E-Mail: a.lux@exxcellent.de)
ist Gesellschafter und Projektleiter von eXXcellent solutions in Ulm. Hier ist er neben seiner Projektleitertätigkeit verantwortlich für den Softwareentwicklungsprozess. Aus seiner langjährigen Tätigkeit in den unterschiedlichsten Projekten konnte er sehr viel Erfahrung, vor allem im Bereich Requirements Engineering, sammeln.

Model Driven Requirements Engineering – Verantwortung im Wandel

Bisher war alles ganz einfach: Das Requirements Engineering spezifiziert die Anforderungen, die Softwareentwicklung analysiert die Aufgaben und setzt diese dann in der Anwendung um. Dieser Weg hat sich seit vielen Jahren bewährt, ist aber leider sehr ineffizient und aufwendig. Ein deutlich effizienterer Weg könnte Softwareentwicklung mit modellgetriebener Anforderungsanalyse (Model Driven Requirements Engineering – MDRE) sein. Hierbei werden die Anforderungen so beschrieben, dass sie maschinenlesbar – in Form eines Modells – direkt von der zu erstellenden Anwendung verarbeitet werden können. In diesem Artikel wird analysiert, wie sich dieser Ansatz auf den Entwicklungsprozess und die Verantwortlichkeiten im Projekt auswirkt.

Requirements Engineering (RE) ist ein absolut notwendiges, sehr aufwendiges und somit auch teures Vorgehen in der Softwareentwicklung. Deutlich einfacher wäre es natürlich, wenn alle Beteiligten wüssten, was, wie und wofür etwas gemacht werden muss. Da dies aber so gut wie nie der Fall ist, ist es notwendig, Anforderungen an ein neues Softwaresystem oder Anpassungen an ein solches zu sammeln, aufzubereiten, zu hinterfragen und auf Machbarkeit hin zu untersuchen. Requirements Engineering ist somit die notwendige Vorarbeit für die Softwareentwicklung. Um den Prozess zu optimieren, gibt es seit geraumer Zeit Bestrebungen, die im RE entstehenden Artefakte (Prozesse, Strukturen, Definitionen) so zu erfassen, dass diese von der Softwareentwicklung in möglichst großem Umfang weiterverwendet werden können.

Genau hier erleben wir im Requirements Engineering ein völlig neues Paradigma. Im klassischen RE wurden fachliche Regeln oder Abläufe vom Requirements Engineer bisher sehr oft in Prosa erfasst. Zu einem späteren Zeitpunkt wurden diese von einem Analysten in belastbare Regelwerke umformuliert und dann vom Softwareentwickler auskodiert.

Mit modellgetriebener Anforderungsanalyse (Model Driven Requirements Engineering – MDRE) lassen sich die Artefakte des Requirements Engineerings direkt und ohne weitere Nachbearbeitung oder Detaillierung in der Softwareentwicklung einsetzen. Diese direkte Wiederverwendung schafft eine völlig neue Situation der Verantwortlichkeiten der Rollen, da die Inhalte aus dem Requirements Engineering nicht mehr zwingend vor ihrem Einsatz in der Softwarelösung nachanalysiert werden müssen. Um dieses fehlende prüfende Organ zu ersetzen, sind neue Maßnahmen im Softwareerstellungsprozess gefordert, um auch langfristig erfolgreiche Lösungen erstellen zu können.

Klassisches RE in der Softwareentwicklung

Ziel des Requirements Engineering ist es, Anforderungen zu ermitteln, zu prüfen und zu beschreiben, um diese in einem Softwaresystem umzusetzen. Alle drei Tätigkeitsfelder können basierend auf bewährten Methoden durchgeführt werden.

Wir beschränken uns in diesem Artikel auf das Thema „Beschreiben und Weiterverarbeiten von Anforderungen“, da dies die Schnittstelle des Requirements Engineerings zur Softwareentwicklung ist und wir hier die Potenziale besser nutzen wollen.

Beschreiben von Anforderungen

Im klassischen Requirements Engineering Prozess werden die Anforderungen sehr häufig textuell beschrieben, entweder in reinen Textverarbeitungen, sehr oft aber auch unter Verwendung von RE Tools. Diese sind i. d. R. datenbankbasiert und erlauben das Erfassen und Strukturieren von Requirements nach vordefinierten Schablonen. In beiden Fällen sind die Anforderungen überwiegend als natürlich sprachlicher Text erfasst, der in Einzelfällen mit grafischen Schaubildern untermauert wird.

In seltenen Fällen wird eine standardisierte formale Sprache wie z. B. UML (siehe [UML]) oder BPMN (siehe [BPMN]) ver-

wendet. Doch auch in diesen Fällen handelt es sich bei den erfassten Diagrammen sehr oft um Informationssackgassen, was bedeutet, dass die Informationen zwar basierend auf einer formalisierten Standardsprache erfasst wurden, diese aber eher als Grafik denn als Modell verstanden werden und in den seltensten Fällen automatisiert weiterverarbeitet werden.

Weiterverarbeitung von Anforderungen
Die weitere Verarbeitung der Definitionen aus dem klassischen Requirements Engineering folgt typischerweise immer demselben Schema: Die Anforderungen werden von einem Softwareentwickler oder Analysten hinterfragt und bewertet. Dieser nachgelagerten Analyse ist ein überaus bedeutender Wert zuzurechnen, denn hierbei werden die im Vorfeld beschriebenen Anforderungsdefinitionen auf Machbarkeit, Plausibilität und (soweit möglich) auf Korrektheit überprüft.

Kommt es hierbei zu Missverständnissen oder lassen die Definitionen Mehrdeutigkeiten zu, werden diese mit dem Requirements Engineer diskutiert und bereinigt. Danach wird aus den Anforderungsdefinitionen vom Softwareentwickler die eigentliche Funktionalität geformt, was konkret bedeutet, dass er Sourcecode, Konfigurationen, Datenbankeinträge usw. erstellt. Am Ende überprüfen Tester die neue oder geänderte Funktionalität gegenüber den Definitionen aus dem Requirements Engineering und weisen so – im Idealfall – die Korrektheit und Vollständigkeit der umgesetzten Anforderungen nach.

Unterschied klassisches RE zu MDRE

Das oben beschriebene Vorgehen wird in vielen Projekten so oder in ähnlicher Form umgesetzt und ist leider alles andere als effizient, da häufig Inhalte aus dem Requirements Engineering lediglich in eine andere Form (z. B. Code) gebracht werden, ohne diese signifikant mit Details anzureichern.

Der Grund hierfür ist, dass in der Softwareentwicklung für fachliche Aufgaben sehr schnell Abstraktionen implementiert werden und der für die spezifische Fachlichkeit benötigte Code in vielen Fällen dem Spezifikationstext sehr ähnelt.

Beispiel 1 – klassisches RE:

1. Der Requirements Engineer beschreibt textuell die Regeln des Dialogflusses zu einem Hilfeassistenten.
2. Der Analyst entwirft dazu die Struktur des Controllers, z. B. in Form eines Statechart- Diagramms.
3. Der Entwickler nimmt das Diagramm und kodiert basierend auf dem Diagramm den UI-Controller (sehr oft basierend auf einer allgemeinen Controllerfunktionalität).
4. Über einen automatisierten Build und Deployment Prozess wird eine testbare Applikation erstellt und auf ein Testsystem deployed.
5. Der Tester nimmt die Requirement-Definition und prüft (ggf. automatisiert), ob alle Anforderungen abgebildet wurden und funktionieren.

Von der Anforderungsanalyse bis zum abnehmenden Test sind mindestens vier Rollen und mindestens ebenso viele Arbeitspakete beteiligt. Wenn sich die Anforderungen in einer späteren Iteration ändern, muss die gesamte Prozesskette für dieses Feature von neuem durchlaufen werden. Dieses klassische Vorgehen lässt sich mithilfe des MDRE deutlich effektiver gestalten.

Der Kern von MDRE basiert auf der direkten Weiterverarbeitung von Anforderungsdefinitionen bei der Softwareentwicklung. Dies bedeutet konkret, dass bereits beim Requirements Engineering Inhalte so erfasst werden, dass diese direkt in die Softwareentwicklung eingebaut werden können oder dass aus den RE-Artefakten über Generatoren die entsprechenden Implementierungs-Artefakte erzeugt werden. Alternativ und prinzipiell gleichwertig zu generativen Ansätzen können natürlich auch generische Ansätze verfolgt werden, bei denen das fachliche Modell zur Laufzeit von einer Runtime interpretiert wird.

Beide Vorgehen basieren auf der Annahme, dass eine Softwarelösung für eine bestimmte fachliche Domäne erstellt wird und dass es eine begrenzte Anzahl fachlicher Grundszenarien gibt. Eine Verwaltungslösung im Krankenversicherungsumfeld wird sich z. B. nur in den wenigsten Fällen mit hardwarenahen Realtimeanforderungen auseinandersetzten müssen, sondern sich auf fachliche Grundszenarien wie Stammdatenerfassung, Abrechnungen, Lastschriften, Reporting u. ä. konzentrieren – dies aber natürlich in allen möglichen Facetten.

Unter dieser Annahme kann die fachliche Problembeschreibung als abstrakte Sicht auf die konkrete Lösung verstanden werden. Und genau dies ist der Kern von modellgetriebener Softwareentwicklung. Die Kunst besteht nun „nur“ noch darin, eine abstrakte Sicht zu identifizieren, die sowohl vom Requirements Engineer fachlich als auch vom Softwareentwickler technisch richtig interpretiert wird und diese auch in geeigneter Form (z. B. als grafisches oder textuelles Modell) den Beteiligten zur Bearbeitung zur Verfügung zu stellen. Erreicht werden kann dies durch das Erstellen einer eigenen domänenspezifischen Sprache (siehe [DSL]).

Beispiel 2 – MDRE:
Hierzu greifen wir das Beispiel 1 nochmals auf.

Vorbedingung: Alle Dialoge des Systems wurden als eigenständige Modellelemente (z. B. als UI-Dialog-Modelle (siehe [ORCH]) oder als UML UseCases mit einem speziellen Stereotyp, z. B. Assistenten-Dialog erfasst.

1. Einmalig für alle Requirements eines Typs:
   a. Definition der DSL für den Requirements Engineer
   b. Implementierung des dazugehörigen Frameworks
2. Der Requirements Engineer beschreibt modellbasiert die Regeln des Dialogflusses zu einem Hilfeassistenten.
3. *weiter Analyse entfällt.*
4. *konkrete Implementierung entfällt.*
5. Über einen automatisierten Build- und Deployment-Prozess wird eine testbare Applikation erstellt und auf ein Testsystem deployed.
6. Der Tester nimmt die Requirement-Definition und prüft (ggf. automatisiert), ob alle Anforderungen abgebildet wurden und funktionieren.

Warum modellgetriebenes RE?
Der Grundgedanke hierbei ist recht einfach und einleuchtend: Definitionen, die im Requirements Engineering-Prozess festgelegt werden, werden exakt so wie beschrieben im System angewendet. Hierzu sind keine Manipulationen mehr notwendig. Durch dieses Vorgehen wird verhindert, dass bei der manuellen Transformation Inhalte falsch übernommen oder gar vergessen werden. Das einzige relevante Modell ist das Fachliche.

Dieses Vorgehen lässt sich natürlich praktischerweise nicht auf alle Features des

Zielsystems ausdehnen und das ist auch nicht der Anspruch, aber bei ausreichend großen und homogenen Systemen kann man durch dieses Vorgehen eine beträchtliche Anzahl von Geschäftsregeln abdecken. Um dies zu erreichen, ist es oft weder notwendig noch hilfreich, auf bestehende Modellierungssprachen zurückzugreifen. Diese sind schon aufgrund ihres Selbstzwecks so konzipiert, dass sie für möglichst viele Einsatzgebiete angewendet werden können und so allgemein gehalten, dass sie viel zu viele Freiheitsgrade zulassen, somit leider mehrdeutig sein können. Viel effizienter können Sprachen eingesetzt werden, die auf die jeweilige Fachdomäne und den verwendeten Generator zugeschnitten sind (Domain Specific Languages, siehe [DSL]).

Konsequenzen für das RE

Was bedeutet dies nun für die Beteiligten? Mutiert der Requirements Engineer zum Softwareentwickler? Nein, das sollte auf keinen Fall notwendig sein. Der Requirements Engineer (als Rolle mit keinem bzw. wenig Wissen über Implementierungsdetails, wohl aber mit Wissen über Funktionalität und Machbarkeit) befasst sich auch weiterhin mit seinen fachlichen und nicht mit technischen Aufgaben.

Dies kann allerdings nur dann funktionieren, wenn das Modell, mit dem er arbeitet, ebenfalls fachlich motiviert ist. Technische Details haben in fachlichen Modellen erstmal nichts zu suchen. Falls es fachlich erforderlich ist, muss es natürlich trotzdem die Möglichkeit geben, technische Varianten (z. B. unterschiedliche Controllertypen) in den Modellen auszudrücken. Dies wird erreicht, indem die (fachlichen) Modellelemente vom Requirements Engineer (fachlich) stereotypisiert oder getagged werden. Die Umsetzung der Tags oder Stereotypen in das entsprechende technische Verhalten wird vom Generator oder der Laufzeitumgebung übernommen.

Dass es sich im Beispiel 2 überhaupt um einen Assistenten handelt, muss nicht zwingend am Modelltyp des Diagramms festgelegt werden. Dies könnte z. B. auch über ein (fachliches) Flag (z. B. ControllerTyp = „Assistent“) am Diagramm definiert werden. Was aber vermieden werden sollte, ist, dass am Diagramm die Controllerklasse referenziert wird. Diese Information ist rein technisch und kann vom Requirements Engineer weder getroffen noch hinterfragt werden. Diese Information sollte besser

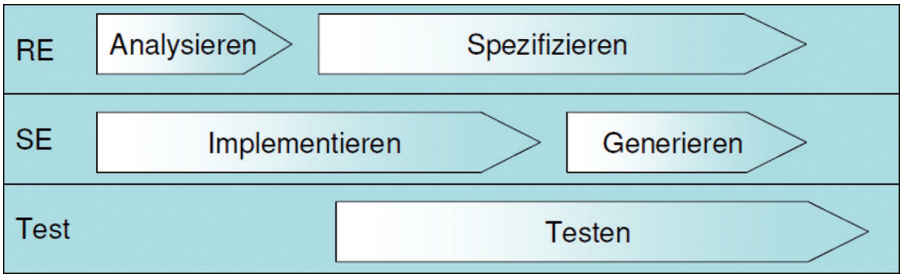

Abb. 1: Zusammenspiel DSL und Frameworks

z. B. als Mapping im Zuständigkeitsbereich der Softwareentwicklung liegen.

Die Trennung von fachlichen Varianten und deren technischer Umsetzung ist immens wichtig, da fachliche Modelle typischerweise länger leben als deren technisches Pendant und somit auch Technologiewechsel überstehen müssen. Es ist essenziell, dass der Requirements Engineer im Detail versteht, wie „seine“ Modellierungssprache funktioniert – sprich: welche sprachlichen Möglichkeiten er hat und welche Auswirkungen diese in der konkreten Anwendung haben. Wird das fachliche Modell mit technischen Details angereichert, ist dies oft nicht mehr der Fall, da der Requirements Engineer die technische Welt nicht in der geforderten Tiefe versteht.

Prüfen der Requirements auf Umsetzbarkeit

Anhand der bisher genannten Punkte kann leicht nachvollzogen werden, dass die Weiterverarbeitung von modellbasierten Requirements in einer Anwendung sehr effizient funktionieren kann. Dies hat allerdings zur Konsequenz, dass sich die Softwareentwickler deutlich mehr um Frameworks und Generatoren kümmern als um die eigentliche Umsetzung fachlicher Regeln in Code. Dies geht sogar soweit, dass sich Analysten und Softwareentwickler inhaltlich mit den konkreten Regeln überhaupt nicht mehr auseinandersetzen, sondern „nur“ drauf bedacht sind, dass die Vielzahl der Varianten an Regeln durch „ihr“ Framework auch abgedeckt werden kann.

Eine Analyse der eigentlichen fachlichen Regeln auf mögliche Inkonsistenzen oder Unvollständigkeiten, wie sie aus dem klassischen Requirements Engineering bekannt sind, wird in vielen Fällen bei MDRE-basierten Projekten nicht mehr stattfinden. Wird an dieser Stelle im Projekt ein Problem (z. B. systematische mangelnde Pfadabdeckung bei Abläufen) vermutet, so muss mit geeigneten Mittel aktiv dagegen vorgegangen werden. Geeignete Mittel können z. B. die zusätzliche Hinterlegung exemplarischer Szenarien oder gemeinsame Workshops mit RE, Analysten, Softwareentwicklern und Tester sein. Hier werden neben der reinen Vermittlung fachlicher Inhalte vor allem die Softwareentwickler und Tester für fachliche Probleme sensibilisiert.

RE-Modelle sind Teil der Software

Überaus wichtig für ein modellgetriebenes Requirements Engineering in der Applikationsentwicklung ist die sehr enge Bindung zwischen fachlichen Modellen aus dem RE und dem Applikationscode. Dies bedeutet, dass es immer eine eindeutige Zuordnung von fachlichen Modellen zu Applikationsreleases geben muss, denn die fachlichen Modelle SIND die Geschäftslogik und nicht wie bisher lediglich die Basis, auf der diese erstellt werden. Änderungen im fachlichen Modell sind quasi eine funktionale Änderung der Anwendung.

Da in der Regel sowohl das fachliche Modell des RE als auch der Applikationscode iterativ und inkrementell erstellt werden, bedeutet dies in letzter Konsequenz, dass die RE-Modelle zusammen mit dem Anwendungscode versioniert werden müssen. Nur so lassen sich auch rückwirkend Versionsstände erstellen, bei denen Modelle und Applikationscode zueinander kompatibel sind. Nur wenn beide Stände zueinander passen, erhält man ein funktionierendes Ganzes.

Meins und seins …

Sätze wie „Ich weiß, dass in der Anwendung seit Stand X die Funktion Y nicht mehr geht, aber DIE (Req. Eng.) müssen

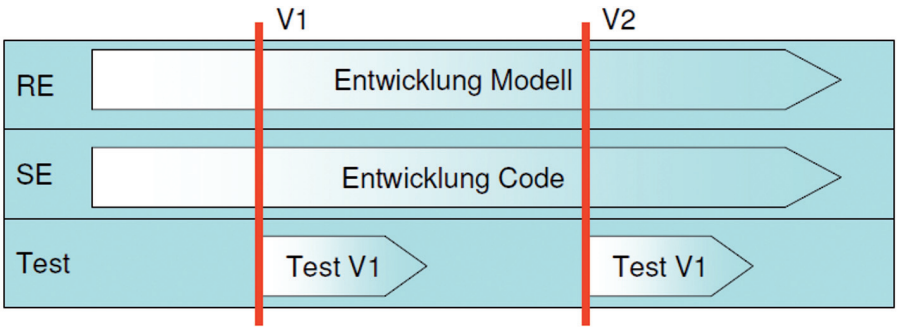

Abb. 2: Synchronisation Modell und Code".

halt aufpassen, was sie spezifizieren …" (Implementierer) kommen in ähnlicher Form bei Projekten, in denen mit MDRE gearbeitet wird, fast zwangsläufig vor. Und genau hier liegt der Unterschied zum herkömmlichen Requirements Engineering.

Bisher war das Requirements Engineering eine Vorleistung zur Softwareerstellung. Mit MDRE wird die Kopplung zwischen Requirements Engineering und Implementierung deutlich enger. Änderungen in der Spezifikation von Anforderungen schlagen direkt auf die Anwendung durch. Der Requirements Engineer entwickelt sich vom „Vorarbeiter" zum „Mitarbeiter". Die Erstellung von fachlichen Modellen wird hierbei genauso zum Bestandteil der Zielanwendung wie die Artefakte aus der Implementierung.

Und genauso auf der Requirementsseite. Die Denkweise *„… das reicht, tief genug spezifiziert. Den Rest werden die in der Implementierung schon hinkriegen …"* kann bei MDRE-basierten Applikationsteilen nicht länger aufrechterhalten werden. Die fachlichen Modelle sind genauso detailreich, wie sie es sein müssen, damit die Frameworks aus der Softwareentwicklung die geforderte Funktionalität umsetzen können.

Testen

Bei MDRE-Projekten ist es immens wichtig, fachliche und technische Tests zu trennen. Tests auf Softwareentwicklerseite fokussieren sich vermehrt auf Funktionalitäten der verwendeten Frameworks. Logiken aus Modellen des RE werden typischerweise von der Softwareentwicklung nicht so ohne weiteres über (automatisierte) Testfälle abgedeckt. Der Grund hierfür ist einfach und nachvollziehbar: Automatisierte Tests werden von den Softwareentwicklern eingesetzt, um die korrekte Funktionsweise der von ihnen kodierten Funktionalität über Regressionstests sicherzustellen. Fehlgeschlagene Tests indizieren eine fehlerhafte Implementierung. Da die eigentliche fachliche Logik nicht aus „der Feder" der Entwickler stammt, sondern von Seite RE über Modelle in die Anwendung einfließt, würde z. B. der Test zum korrekten Ablauf des Assistenten aus Beispiel 2 bei einer geänderten fachlichen Ablauflogik fehlschlagen und auf eine fehlerhafte Komponente hinweisen. Dies wäre hier aber nicht der Fall.

Eine Möglichkeit, das Zuständigkeitsproblem beim Testen zu lösen, wäre z. B. eine eigene Rolle im Projekt zu platzieren, die sich rein um die Erstellung automatisierter FACHLICHER Tests kümmert. Aus Gründen der Effizienz sollte diese Rolle ebenfalls vom Requirements Engineer in Personalunion übernommen werden. Da diese aber leider sehr of bereits mit dem Thema RE zeitlich ausgelastet sind, werden sie das in den seltensten Fällen auch leisten können.

Fachliche Tester aber haben ein systematisches Problem. Weder spezifizieren sie die Anforderungen noch implementieren sie deren Funktionalität. Wie also sollen sie das fachliche Know-how zur Erstellung der Tests aufbauen? Nur aus den Dokumenten und Modellen wird das nicht in ausreichendem Maße möglich sein. Um dieses Wissen aufzubauen, müssen die Tester „zwischen den Zeilen" lesen können. Dies funktioniert erfahrungsgemäß am besten, wenn die Tester in den Übergabeworkshops RE→ SW-Entwicklung beteiligt sind. Denn hier (und vermutlich nur hier) werden die fachlichen Themen inhaltlich diskutiert.

Erfolgsfaktor Hirn

Maßgebend für den erfolgreichen Einsatz von MDRE im Projekt ist die passende Definition der fachlichen Modellsprache. Über diese müssen sich alle fachlichen Varianten ausdrücken lassen. Es sollte nicht notwendig sein, dass der Requirements Engineer auf technische Details eingehen muss, die nicht tatsächlich auch fachlich relevant sind.

Beispiel 3 :

Hierzu greifen wir das Beispiel 2 nochmals auf: *„Der Requirements Engineer beschreibt modellbasiert die Regeln des Dialogflusses zu einem Hilfeassistenten."*

In der Praxis ist das vermutlich nicht ganz so einfach. Es könnte sich z. B. wie folgt darstellen:

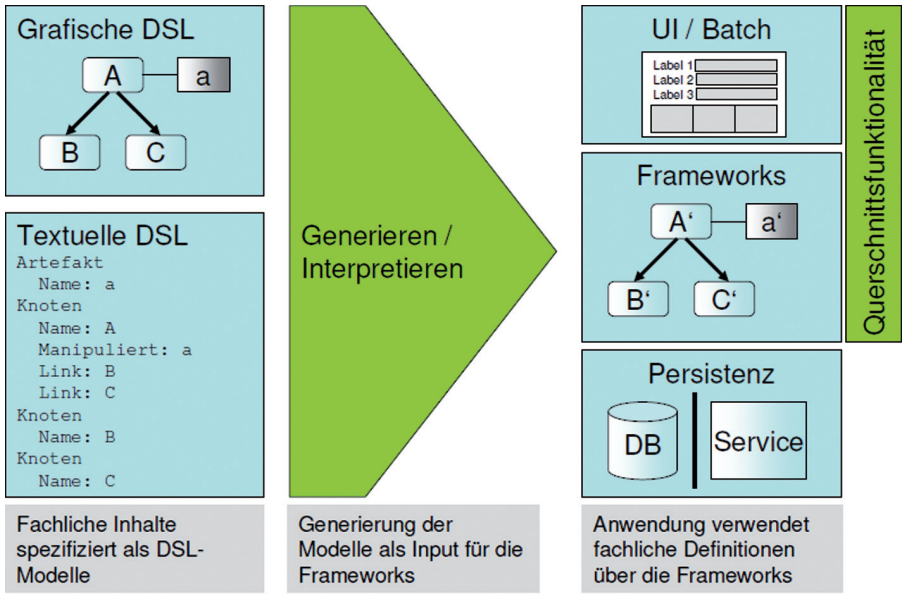

Abb. 3: Phasen bei MDRE

Der Requirements Engineer erfasst in einem grafischen Editor, basierend auf einer für diese Problemdomäne erstellten DSL (siehe [DSL]), den Ablauf für den Assistenten. Im Assistenten-Modelleditor hinterlegt er die Bedingungen zur Erreichung des nächsten Dialogschrittes in Form eines OCL-Ausdrucks (siehe [OCL]).

Der Requirements Engineer arbeitet in diesem Beispiel mit Dialogelementen und mit OCL als formaler Sprache zur Formulierung von "weiter"-Bedingungen. Trotzdem darf man den MDRE-Ansatz im Projekt nicht allzu theoretisch und absolutistisch sehen. Abhängig von der Problemstellung lassen sich die fachlichen Regeln mehr oder weniger gut in eine Modellierungssprache packen. Auch wenn es anfangs den Anschein hat, dass sich eine Problemstellung gut in einer Modellierungssprache abbilden lässt, zeigt die Erfahrung, dass diese im Projekt nicht immer durchgehalten werden kann. Die Abbildung von Ausnahmen in einer Modellierungssprache kann sehr schnell dazu führen, dass die Sprache sehr unübersichtlich und unhandlich wird und das originäre Ziel, mehr Qualität und Effizienz im Prozess zu etablieren, zum Scheitern verurteilt wird. Besser ist es hier, auf Applikationsseite (Framework/Generator) Möglichkeiten zu schaffen, Ausnahmen bei Bedarf auszukodieren.

Wie im Beispiel 3 beschrieben, erstellt der Requirements Engineer mittels OCL die Regeln, wann der „Weiter"-Button aktiviert wird. Es kann natürlich auch Fälle geben, in denen sich die fachliche Logik nicht mehr oder nur sehr umständlich (also auch unverständlich) über OCL ausdrücken lässt. In diesen Fällen werden die Regeln vom Requirements Engineer textuell erfasst und vom Entwickler auscodiert. Hierzu bedarf es eines Flags (z. B. „kodierte Regel") im Modell, auf das ein Generator reagieren kann. Diese Fälle sollten jedoch zu den absoluten Ausnahmen gehören, da hier die fachliche Beschreibung und die Implementierung natürlich wieder differieren können.

Fazit

Wie bei allen neuen Vorgehensweisen ist es wichtig, eine Akzeptanz für das neue Vorgehen zu schaffen und die beteiligten Rollen nicht zu überfordern. Wer sich noch nie mit formalen Sprachen auseinandergesetzt und die Anforderungen bisher ausschließlich in Prosa verfasst hat, wird sich auf den ersten Blick sehr eingeschränkt vorkommen, da er nicht wie in der Vergangenheit alle Belange in einer willkürlichen Vielfalt ausdrücken kann, sondern sich bei der Formulierung der Anforderungen auf eine formale Sprache stützen soll. Aber genau darum geht es beim MDRE. Die willkürliche Vielfalt soll auf das notwendige Maß reduziert und so formuliert werden, dass die Ergebnisse des Requirements Engineering ohne weitere manuelle Aufbereitung für das Zielsystem verwendet werden können.

Die richtige Sprache und deren Anwendung ist das A und O. Der Requirements Engineer muss in der Lage sein, sich sicher in „seiner" Welt der grafischen oder textuellen Modelle bewegen zu können.

Für MDRE ist es überaus wichtig, dass alle Beteiligten sich ihrer neuen Rolle und dem Einsatz der von ihnen produzierten Artefakte vollumfänglich bewusst sind. Dies erfordert ein neues Denken im Requirements Engineering mit der Konsequenz von deutlich mehr Effizienz und Qualität im gesamten Softwareentwicklungsprozess. ■

Referenzen

[DSL]: http://de.wikipedia.org/wiki/Dom%C3%A4nenspezifische_Sprache
[ORCH]: http://www.exxcellent.de/leistungen/orchideo/orchideo_ui_designer.html
[UML]: http://www.uml.org/
[OCL]: http://www.exxcellent.de/download/OS_01.2005_S.55-58.pdf
[BPMN]: http://www.bpmn.org/