mehr zum thema:
www.scandlines.de

von markus reinhold,
rudolf hauber,
dieter wagner

GROSSE SYSTEM IN DEN GRIFF BEKOMMEN

Für die Entwicklung großer Systeme liefern viele Entwicklungsprozesse nur ungenügend Hilfestellung. Ein unternehmensspezifisch angepasstes und optimiertes Vorgehensmodell muss daher folgende Aspekte integrieren: professionelle Anforderungserfassung und -verwaltung, moderne Modellierungstechniken (wie die UML), Werkzeugunterstützung und ein funktionsfähiges Konfigurationsmanagement. Das alles sind Schlüsselfaktoren für den Projekterfolg. In dem Artikel werden die Erfahrungen mit einer entsprechenden Methodik inklusive Werkzeugumsetzung dargestellt.

Die Firma Lenkflugkörper Systeme GmbH ist Teil des EADS-Konzerns und befasst sich seit vielen Jahren erfolgreich mit der Erstellung von Lenkflugkörpern. Derartige Systementwicklungen zeichnen sich durch eine hohe Komplexität aus. Die Anwendungsentwicklung ist nur ein Teil der zu lösenden Aufgabe. Die Entwicklung unterschiedlicher Hardwarekomponenten sowie die Integration verschiedener Anwendungssysteme, Betriebssysteme und Netzwerktechnologien bilden eine weitere Komplexitätsebene. Aufgrund der Konzernstruktur kommt es häufig vor, dass innerhalb eines Projekts verschiedene konzerninterne Unternehmen gemeinsam an einem Projekt mitwirken. Dieser Sachverhalt muss natürlich bei der Vorgehensweise berücksichtigt werden.

Speziell die hohe Komplexität der Projekte und die daraus folgende Informationsflut ist nur mit geeigneten Methoden zu meistern. Dateninkonsistenz sowie redundante Informationen müssen vermieden werden. Aber auch externe Einflüsse — z. B. Änderungswünsche der Kunden — müssen mit den gleichen Bearbeitungsmethoden bewältigt werden. Die grundsätzliche Lösung vieler dieser Probleme ist durchaus bekannt, nur die konkrete Umsetzung ist häufig nicht so leicht zu erreichen, da zuerst Berührungsängste mit neuen Methoden und Vorgehensweisen abgebaut werden müssen. In Zusammenarbeit mit zwei externen Firmen wurde ein Vorgehen einschließlich Werkzeugumsetzung entwickelt, das all diesen Rahmenbedingungen Rechnung trägt.

Lösungsbausteine

Reine Softwareentwicklungsprozesse sind der hier vorliegenden Komplexität kaum gewachsen (vgl. [Rei00]). Aus diesem Grund wurde ein angepasstes Vorgehensmodell auf Basis des international anerkannten Standardvorgehensmodells V-Modell'97 ([VM97]) gewählt und weiterentwickelt. In diesem Vorgehensmodell sind der erweiterte Fokus einer *System*erstellung — im Gegensatz zur reinen *Software*erstellung — sowie die Strukturierung des Vorgehens einschließlich Ergebnisdokumentation auf eine mögliche Vergabe von Systemteilen an Unterauftragnehmern berücksichtigt (vgl. [Oes01]). Im konkreten Fall wurde das Vorgehensmodell mit Anleitungen untermauert, die ein professionelles und werkzeuggestütztes Erfassen, Modellieren und Verwalten von Anforderungen und einer Systemmodellierung unterstützen. Das bedeutet, dass unterschiedliche Techniken (*Requirements-Engineering (RE)* und Modellierung mit der *Unified Modeling Language (UML)*) und dafür verfügbare Werkzeuge zum Einsatz kamen (vgl. [Rei97]). Mit Hilfe einer integrierten Werkzeugumgebung entstand eine Art virtuelle Projektdatenbank, die alle Entwicklungsinformationen beinhaltet, diese über Änderungen hinweg konsistent hält und eine automatische Generierung von Dokumenten ermöglicht.

Requirements-Engineering

RE umfasst die systematische ingenieurmäßige Anforderungsanalyse sowie Maßnahmen, welche die Anforderungsanalyse und die weitere Verwendung der Anforderungen unterstützen (vgl. [Rup01] und [Rob01]). Die Grundidee ist, dass jedes System durch Kunden- bzw. Anwenderforderungen bestimmt ist, zu denen weitere Anforderungen hinzukommen, welche die Realisierung der Anwenderforderungen beeinflussen (Standards, Wiederverwendung, externe Beschaffung von Komponenten, gesetzliche Rahmenbedingungen etc.).

Um große Systeme entwickeln zu können, ist der Ansatz „Teile und Herrsche" entscheidend. Jedes System kann in kleinere, weniger komplexe Subsysteme, die sich einfacher realisieren lassen, zerlegt werden. Jedes dieser Subsysteme erfüllt Teile der Gesamtanforderung, produziert durch seine Existenz jedoch auch eigene spezifische (Schnittstellen-)Anforderungen. Um

die autoren

Markus Reinhold (E-Mail: reinhold@cocoo.de) ist Geschäftsführer der Firma CoCOO, die auf Schulung und Beratung für objektorientiertes Software-Engineering und Vorgehensmodelle spezialisiert ist.

Dr. Rudolf Hauber (E-Mail: RHauber@ka-muc.de) arbeitet seit fünf Jahren bei der Kölsch & Altmann Software & Management Consulting GmbH im Bereich Schulung und Beratung.

Dieter Wagner (E-Mail: dieter.wagner@lfk.dasa.de) ist Modulteamleiter bei der Firma LFK GmbH und arbeitet gemeinsam mit seinem Kollegen Thomas Rittel (E-Mail: thomas.rittel@lfk.dasa.de) an der Entwicklung von Lenkflugkörpern

die Gesamtanforderungen zu realisieren, wird eine gewisse Systemarchitektur gewählt, welche die Zerlegung des Gesamtsystems in Subsysteme beschreibt. In die Systemarchitektur gehen die Anforderungen aller beteiligter *Stakeholder* sowie technische Überlegungen und Erfahrungen ein. Eine Systembeschreibung setzt sich somit aus einem Anforderungs- und einem Architekturteil zusammen (**siehe Abb. 1**). Da jedes Subsystem wiederum als System betrachtet werden kann, gilt diese Aussage auf jeder Ebene der Systemzerlegung. Derartig wiederholbare Denkmodelle sind ein wichtiges Instrument zur Komplexitätsbewältigung. Die Anforderungen an die einzelnen Subsysteme leiten sich über mehrere Ebenen aus den Gesamtanforderungen und der gewählten Architektur ab. Wichtig ist hierbei die spätere Nachverfolgbarkeit der Schritte, d. h. es muss *Traceability* sichergestellt werden.

Anforderungsarten

Im Laufe der Projektarbeit wurde sehr schnell klar, dass es unterschiedliche Anforderungsarten zu unterscheiden gilt. Primär ist jedes System in seinem Verhalten durch seine *funktionalen Anforderungen* bestimmt. Für deren Erfassung und Dokumentation hat sich eine Technik bewährt, welche die Kundenperspektive wieder stärker in der Vordergrund stellt und die somit zu einer stärkeren Kundenintegration beiträgt. Gemeint sind sogenannte *Use-Cases* (Anwendungsfälle), die im Rahmen der UML definiert sind. Funktionale Anforderungen alleine reichen zur Beschreibung eines Systems jedoch nicht aus. Daneben existieren unter anderem Sicherheits-, Qualitäts- und Leistungsanforderungen. Diese *nicht-funktionalen Anforderungen* lassen sich mit einer Art Checkliste systematisch erfragen. Meist ist es möglich, die Anforderungen den Use-Cases zuzuordnen. Hier kommt es jedoch häufig zu einer *n:m*-Beziehung (d. h. eine nicht-funktionale Anforderung bezieht sich auf mehrere Use-Cases und umgekehrt). Wir sprechen hier also von der Notwendigkeit, unterschiedliche Informationen (hier Anforderungen) und deren Beziehung verwalten zu müssen.

In vielen Fällen wird dieses Problem mit Hilfe eines Textverarbeitungssystems angegangen, das die Zusammenhänge jedoch nicht effizient verwalten kann. Aus diesem Grund wurde im vorliegenden Projekt ein *Requirement-Management*-Werkzeug (*RM*-Werkzeug) eingesetzt, das datenbankorientiert arbeitet. Mit Hilfe eines solchen Werkzeuges ist es möglich, Beziehungen der Anforderungen untereinander effizient zu verwalten. De Verbindungen fallen nicht automatisch ab, sondern müssen fortwährend analysiert, verifiziert und aktualisiert werden.

Die Erfassung von Use-Cases, nicht-funktionalen Anforderungen sowie die Darstellung von Architekturentscheidungen erfolgte mit entsprechend spezialisierten Werkzeugen und von verschiedenen Personen (Rollen). Ein ganzheitlicher RE-Ansatz muss also mit anderen Entwicklungsmethoden des System-Engineerings — wie z. B. der UML — kombinierbar sein. Mittels des zu Grunde liegenden Entwicklungsprozesses muss die Konsistenz sichergestellt werden. Maximale Effizienz kann nur dann erreicht werden, wenn auch die eingesetzten Werkzeuge die Durchgängigkeit der Entwicklung unterstützen. Insbesondere die Verknüpfungen aus dem RM-Werkzeug zur funktionalen Spezifikation und zum Architekturdesign im Modellierungswerkzeug sollten fließend und werkzeugtechnisch verfolgbar sein.

Die funktionalen Anforderungen wurden weitestgehend in Form von Use-Cases in der UML erfasst und die nicht-funktionalen Anforderungen als natürlichsprachliche *Requirements* im RM-Werkzeug. Die durch die Systemzerlegung notwendige Detaillierung von Anforderungen ist bezüglich der nicht-funktionalen Anforderungen kein Problem. Use-Cases sind gemäß ihrer UML-Definition jedoch nicht hierarchisierbar. Hier musste mit Hilfe einer Erweiterung der UML-Grundsemantik Abhilfe geschaffen werden, um die Use-Cases auf Systemebene durch Use-Cases für die einzelnen Subsysteme der darunter liegenden Ebene realisieren zu können.

Use-Cases und deren Schnittstellen zur umgebenden Welt (sogenannte *Akteure*) sind Bestandteil des Use-Case-Modells. Dieses besteht aus Use-Case-Diagrammen, welche die Use-Cases, die beteiligten Akteure und die Systemgrenzen zeigen. Die detaillierte Beschreibung eines Use-Cases erfolgt in möglichst einfacher verständlicher Sprache. Die verwendeten Begriffe müssen klar definiert sein und sollten aus der Sprachwelt des Kunden/Anwenders stammen. Die Beschreibung basiert auf „semiformalen" Vorlagen, die eine inhaltliche Strukturierung der Information vorgeben. Die Vorlagen sollten mindestens Abschnitte für

- Vorbedingungen,
- Basisablauf,
- alternative Abläufe,
- Nachbedingungen,
- Erweiterungen und
- sonstige Informationen

enthalten. Auch das Layout (Kapitelnummerierung, Formate usw.) sollte im Vorfeld bedacht werden, um Überraschungen bei der automatischen Dokumentationsgenerierung zu vermeiden. Da sich Anforderungen über den Lebenszyklus bekanntlich ändern (vgl. [Hru00]), muss das gesamte Use-Case-Modell einschließlich Beschreibungen unter Versionsverwaltung stehen. Die Vollständigkeit der beteiligten Akteure (menschliche Benutzer, externe Systeme, Umwelt usw.) und ihrer Kommunikationsbeziehungen zu Use-Cases sind wichtig, da damit später die kompletten System- bzw. Subsystemschnittstellen automatisch ermittelt werden können.

Üblicherweise entstehen auf diese Weise einige Dutzend Use-Cases. Diese können durch eine Gruppierung in UML-Pakete strukturiert werden. In der Realität stehen Use-Cases jedoch nicht isoliert nebeneinander, sondern es gibt Zusammenhänge und Beziehungen zwischen ihnen. Die UML sieht dafür einerseits *«include»*- und *«extend»*-Beziehungen zwischen Use-Cases vor. Übergreifende Zusammenhänge können damit jedoch nicht beschrieben werden. Im vorliegenden Fall wurden hierzu Zustandsdiagramme eingesetzt, die es erlauben, die Betriebszustände und -übergänge eines Systems zu beschreiben. Damit können Sicherheitsaspekte erfasst werden („Wann darf was gemacht werden?"). Geschachtelte Zustände können benutzt werden, um Verfeinerungen von Zuständen in interne Subzustände und ihre Übergänge zu beschreiben. Wenn ein Use-Case innerhalb eines Zustands ausgeführt werden darf, wird dies durch eine interne Aktion dieses Zustands modelliert. Zusammenhänge zwischen Use-Cases mit Hilfe von Zustandsdiagrammen darzustellen, hat sich unter den gegebenen Umständen als machbar und hilfreich erwiesen.

Anforderungen zerlegen

Subsysteme wurden wie eigenständige Systeme betrachtet, um eine Vergabe an Unterauftragnehmer zu ermöglichen. Somit sollten Anforderungen an

Subsysteme ebenso wie eigenständige Systeme mit Use-Cases und nicht-funktionalen Anforderungen in Textform beschrieben werden. Wie kommt man nun zu den Use-Cases der Subsysteme?

Ein Use-Case wird auf Systemebene durch *n* Use-Cases für die einzelnen Subsysteme der direkt darunter liegenden Ebene realisiert. Ein Use-Case teilt sich somit auf mehrere Use-Cases der an seiner Realisierung beteiligten Subsysteme auf (**siehe Abb. 2**). Die Realisierung der Use-Cases und damit die Anforderungszuordnung an die Subsysteme der direkt darunter liegenden Ebene wird über Sequenzdiagramme modelliert. Ein Sequenzdiagramm zeigt für einen konkreten Ablauf eines Use-Cases die Aktionen, welche die beteiligten Subsysteme der direkt darunter liegenden Ebene durchführen, und deren Reihenfolge. Diese Aktionen bzw. Aktionsfolgen stellen Use-Cases für die einzelnen Subsysteme dar. Die beteiligten Subsysteme werden als Objekte in den Sequenzdiagrammen modelliert.

- In einem *ersten Schritt* werden die Subsysteme der direkt nächsten Ebene ermittelt (Systemarchitekturentscheidung).
- Im *zweiten Schritt* werden für jeden Use-Case mehrere Sequenzdiagramme erstellt. Mindestens der Basisablauf des Use-Cases muss in einem Sequenzdiagramm modelliert werden. Für wichtige alternative Abläufe werden weitere Sequenzdiagramme erstellt. Die Use-Cases der Subsysteme ergeben sich somit aus einzelnen Nachrichten oder Nachrichtenfolgen der Sequenzdiagramme der System-Use-Cases. Die Vollständigkeit aller Abläufe (Basisablauf und alternative Abläufe) und die Synchronisation der einzelnen Aktionen eines Use-Cases können durch ein integratives Zustands- oder Aktivitätsdiagramm erreicht werden. Dieses beschreibt die Gesamtheit aller möglichen Abläufe und ihre Zusammenhänge. Insbesondere bei komplexen Ablaufbedingungen oder bei Nebenläufigkeit ist dies sehr hilfreich.
- Im *dritten Schritt* werden die Subsystem-Use-Cases detailliert beschrieben. Für diese werden anschließend Sequenzdiagramme und gegebenenfalls ein zusammenfassendes Aktivitätsdiagramm erstellt. Daraus ergeben sich rekursiv jeweils die Use-Cases der nächst tieferen Ebene (**Abbildung 2**).

Darstellung der Systemarchitektur mit der UML

Die Systemarchitektur beschreibt den statischen Aufbau eines Systems aus Subsystemen inklusive der Zusammenarbeit der Subsysteme. Begleitend zur Anforderungsdetaillierung und -zerlegung muss auch die Systemzerlegung in Subsysteme erfolgen. Dieser statische Aufbau wird in der UML durch Pakete abgebildet. Diese werden mit geeigneten Stereotypen gemäß V-Modell (d. h. «Segment» oder «SW-Einheit») versehen, die eine zusätzliche Semantik in das Modell einbringen. Diese Information kann später für Konsistenzprüfungen und die Dokumentationsgenerierung genutzt werden. Abhängigkeiten zwischen kommunizierenden Subsystemen werden durch UML-Abhängigkeitsbeziehungen modelliert. Entscheidend ist, dass die Abhängigkeiten zwischen den Subsystemen einer Ebene nicht einfach als Wunschbild eingezeichnet werden, sondern dass ein werkzeuggestützter Mechanismus existiert, um aus der tatsächlichen Modellierungssituation heraus zu überprüfen, ob Verstöße gegen die geplante Abhängigkeitsstruktur vorliegen. Diese Prüfung ist nicht einfach durchzuführen, da sich implizite Abhängigkeiten aus sehr vielen unterschiedlichen Gründen ergeben können (Operationsaufrufe, Vererbungshierachien, Benutzung usw.). Wenn Wunsch und Wirklichkeit nicht übereinstimmen, führt dies in späteren Entwicklungsphasen zu erheblichen Problemen, da einzelne Subsysteme nicht unabhängig von anderen entwickelt und getestet werden können.

Bei der Beschreibung der Zusammenarbeit der Subsysteme in der Systemarchitektur wird gleichzeitig die Brücke zur Anforderungsdetaillierung und -zerlegung geschlagen. Zu jedem Use-Case werden — wie oben beschrieben — Sequenzdiagramme erstellt. Diese bilden das integrative Element, da hier die Subsysteme der direkt darunter liegenden Ebene als Objekte auftreten. Einerseits werden Subsysteme also als Pakete dargestellt, andererseits als Instanzen von Klassen. Viele Modellierungswerkzeuge erlauben dies derzeit nicht. Man kann jedoch für jedes Paket eine Klasse gleichen Namens und Stereotyps definieren, die das Subsystem repräsentiert. Somit kann dem Subsystem Verhalten in Form von Operationen zugewiesen werden. Die Nachrichten bzw. Nachrichtenfolgen an Subsysteme in den Sequenzdiagrammen stellen Use-Cases für diese dar. Die Use-Cases der Subsysteme müssen den Nachrichten, die in den Sequenzdiagrammen an sie fließen, entsprechen. Da hinter den Nachrichten Operationen der Subsysteme stehen, entsprechen die Use-Cases letztlich Operationen der Subsysteme. Diese Zusammenhänge sind in einem großen System kaum mehr manuell zu prüfen und sollten daher werkzeuggestützt (mittels Skript) überprüft werden (**siehe Abb. 3**).

Beim Erstellen einer Systemarchitektur ist es wichtig, die Schnittstellen zwischen den Subsystemen zu identifizieren. Ein Subsystem kann mit den systemexternen Akteuren und Subsystemen der gleichen oder höheren Ebene kommunizieren. Für Use-Cases dieses Subsystems sind all diese Kommunikationspartner extern — also Akteure. Dies ist wichtig, um automatisch Schnittstellenbestimmungen für die einzelnen Subsysteme durchführen zu können. Dieses Vorgehen wird rekursiv über alle Subsystemebenen angewandt. Wenn die HW-/SW-Einheitenebene erreicht ist, werden die Use-Cases für die jeweilige Softwareeinheit entsprechend eines beliebigen Softwareentwicklungsprozesses realisiert. Dies wird ebenfalls durch das V-Modell unterstützt, wäre jedoch auch durch Prozessmodelle — wie z. B. den *Rational Unified Process (RUP)* — umsetzbar.

Integration des RM-Werkzeugs mit dem UML-Werkzeug

Anforderungen, die nicht als Use-Cases sinnvoll verwaltbar sind, wurden in einem RM-Werkzeug erfasst, kategorisiert und mit Attributen versehen. Hierbei wurden mehrere Typen (RequisitePro-Klassen, DOORS-Module) von Informationen unterschieden (**siehe Abb. 4**): Kundenanforderungen (UREQ), Systemanforderungen (SYSREQ) und Subsystemanforderungen (SUBREQ) sowie die zugehörigen Nachweise: Akzeptanznachweise (ANW), Systemnachweise (SYSNW) und Subsystemnachweise (SUBNW).

Die verschiedenen Informationen müssen vollständig miteinander verknüpft werden. So kann beispielsweise eine Kundenanforderung „Das System soll unmittelbar reagieren” zu einer Systemanforderung „Systemreaktion muss immer innerhalb von xxx ms erfolgen” werden,

die sich auf mehrere Use-Cases bezieht und damit implizit auf mehrere beteiligte Subsysteme wirkt. Ferner werden Testfälle benötigt, die diese Anforderungen auf System- und Subsystemebene nachweisen. Jede Verknüpfung hat dabei eine spezielle semantische Bedeutung. Die Bedeutung ist für ein wirkungsvolles RM unerlässlich.

Die Vielzahl der Anforderungen an ein komplexes System muss strukturiert werden, damit sie verständlich und kontrollierbar bleiben. Es sollten jedoch keine zu komplizierten Strukturen in der RM-Datenbank geschaffen werden, da ansonsten das Zurechtfinden erschwert und die Dokumentationsgenerierung unflexibel wird. Ferner ist es empfehlenswert, zwischen

- Anforderungen,
- Erläuterungen,
- Überschriften und
- Randbedingungen

zu unterscheiden.

Generell sollten nicht-funktionale Anforderungen genau wie Use-Cases in möglichst einfacher, verständlicher Sprache eindeutig und präzise formuliert werden, damit nicht nur Fachexperten sie verstehen, sondern auch der Anwender/Kunde (vgl. [Rup00]). Ferner müssen Akzeptanzkriterien aufgestellt werden, anhand derer die Anforderungen dem Auftraggeber gegenüber nachweisbar sind. Anforderungen, die nicht nachgewiesen werden können, sind sinnlos. In der ISO 9000 findet sich hierzu folgende Qualitätsdefinition: „Eine Anforderung ist fehlerfrei umgesetzt, wenn das System alle Akzeptanzkriterien erfüllt".

Attribute (z. B. Status, Autor, Quelle/Besitzer, Kosten, Dringlichkeit) sollten verwendet werden, um Anforderungen weiter zu qualifizieren. Sie unterstützen das Projektmanagement, das Konfigurationsmanagement und die Qualitätssicherung, da sie ein zentrales Hilfsmittel für das Filtern und Sortieren von Anforderungen sind. Das setzt jedoch voraus, dass die Attributinformationen ständig gepflegt und fortgeschrieben werden.

Zwischen den erläuterten Anforderungsarten sind folgende Beziehungen möglich (**siehe auch Abb. 5**):

- Use-Cases können durch eine Zerlegung auf mehrere Use-Cases der an ihrer Realisierung beteiligten Subsysteme verweisen.
- Nicht-funktionale Anforderungen können auf Anforderungen oder Use-Cases der gleichen oder einer tieferen Ebene verweisen.
- Randbedingungen können auf nicht-funktionale Anforderungen oder Use-Cases verweisen.
- Leistungskenngrößen können auf nicht-funktionale Anforderungen oder Use-Cases verweisen.

Projektdatenbank

Ein Ergebnis dieses integrierten Vorgehens war es, dass alle entwicklungsrelevanten Informationen in einer zentralen Projektdatenbank verwaltet werden. Diese Informationen stammen aus völlig unterschiedlichen Quellen: aus Kunden- und Firmenanforderungen, aus Rahmenbedingungen, aus der Systemmodellierung mit UML, aus der Qualitätssicherung oder aus dem Projektmanagement. Die Projektdatenbank repräsentiert die Wissensbasis für alle Projektbeteiligten. Alle nötigen Informationen — z. B. Projektmanagement-Informationen und Entwicklungsdokumente — werden aus ihr gewonnen.

Da diese Informationen mit unterschiedlichen Werkzeugen erfasst und verwaltet werden, existiert die zentrale Projektdatenbank nicht als eine reale Datenbank. Sie entsteht durch die durchgängige Integration der beteiligten Werkzeuge. Je besser die Integration realisiert ist, desto geringer sind die Informationsverluste beim Übergang der Werkzeuge. Nur eine durchgängige Verfolgbarkeit der unterschiedlichen Informationselemente (nicht-funktionale Anforderung auf Use-Cases auf Subsysteme und geeignete Testfälle) ermöglicht es, die Komplexität eines großen Systems in den Griff zu bekommen.

Da die Informationen aus unterschiedlichen Quellen und Sichten in die zentrale Projektdatenbank einfließen, besteht die Gefahr, dass Inkonsistenzen entstehen. Ferner ist es ein menschlicher Zug, dass projektspezifische Richtlinien (RM, Modellierung) nicht befolgt werden. Daher ist es entscheidend, die Projektdatenbank — soweit möglich — automatisch auf Inkonsistenzen und Richtlinienverstöße hin zu untersuchen. Dabei sollten Skripte nicht automatisch Änderungen durchführen, sondern den Projektbeteiligten die fehlerhaften Stellen aufzeigen, um in einem kontrollierbaren Änderungsvorgang beseitigt werden zu können.

Verwendete Werkzeuge und projektspezifische Anpassungen

Im konkreten Fall wurde die „Rational Suite Development Studio" für die Anforderungsverwaltung und die UML-Modellierung eingesetzt. Die Projektdatenbank besteht aus:

- der RM-Datenbank „RequisitePro", welche die nicht-funktionalen Anforderungen und Nachweise verwaltet,
- dem UML-Modell, das aus der zentralen „Rational Rose Model"-Datei (mit der Namenserweiterung *.mdl) und den ausgelagerten „Controled Units" (Namenserweiterung *.cat) für die einzelnen Subsysteme besteht,
- den detaillierten Beschreibungen der Use-Cases in Microsoft Word und
- den projektbegleitenden Informationen (Projekthandbuch, Planungsdokumenten, KM-, QS-Plan, Projektstrukturplan, usw.).

Das Zusammenspiel der Werkzeuge wurde einerseits durch das integrierte Use-Case-Management von RequisitePro (analog Rose-DOORS-Link) erreicht, das es erlaubt, Anforderungen mit Rose-Use-Cases, -Paketen und -Klassen zu verknüpfen. Andererseits benutzt der Dokumentengenerator („SoDA") die Informationen aus der RM-Datenbank, dem UML-Werkzeug und den WinWord-Dateien.

Es sind jedoch zahlreiche Erweiterungen der Werkzeuge nötig, um die gewünschte Funktionalität zu erreichen:

- Die V-Modell-Terminologie wird über die Verwendung von Stereotypen unterstützt.
- Eine V-Modell-Konformität wird durch den Einsatz von Skripten erreicht, die das UML-Modell auf Konformität zu einer entsprechenden Modellierungsrichtlinie hin prüfen (Sind entsprechende Stereotypen vergeben? Gibt es Sequenzdiagramme für Use-Cases? etc.)
- Um den zusätzlichen Anforderungen einer Systemmodellierung (Hardwareanforderungen) gerecht zu werden, wurden neue *tagged values* eingeführt (z. B. Informationen über Frequenz, *Totzeit*, *Jitter*).
- Zur automatischen Erzeugung der V-Modell-Entwicklungsdokumente (Afo, TAnf, SysArc, SSB...) werden geeignete Dokumentationsvorlagen erzeugt.

„Lessons learned"

Welche Erfahrungen wurden mit den beschriebenen Vorgehensweisen, Methoden und Werkzeuge gewonnen? Von zentraler Bedeutung für den Projekterfolg ist, dass ein permanentes System-Engineering durchgeführt wird, das allen Beteiligten bekannt ist und von allen gelebt wird. Da sich der Projektplan und die Anforderungen ändern, neue Risiken ersichtlich werden usw., müssen alle Informationen gepflegt, aktualisiert und harmonisiert werden. Daher müssen auch alle entsprechenden Datenquellen und Dokumente unter Versionsverwaltung stehen.

Ein weiterer wichtiger Punkt war die frühe und durchgehende Einbindung von Spezialisten für die Kernlösungsbausteine V-Modell in Verbindung mit UML und Requirement-Engineering sowie die Umsetzung in den beteiligten Werkzeugen. Dies ermöglichte es, die neuen Prozesse, Methoden und Werkzeuge von Anfang an zielgerichtet und effizient einzusetzen. Nicht zu unterschätzen war die von den Spezialisten geleistete Überzeugungsarbeit gegenüber dem Management bezüglich der neuen Prozesse, Methoden und Werkzeuge. Nur wenn die Gesamtheit der Neuerungen vom Management mitgetragen wird, ist der Erfolg möglich.

Die Vorgehensweise hat sich in diesem Projekt hervorragend bewährt. Die heute ersichtlichen Vorteile sind auch über den Lebenszyklus des Produkts hinaus betrachtet für zukünftige Produktlinien signifikant — ganz abgesehen von dem persönlichen Gewinn jedes Mitarbeiters und damit des Unternehmens. ■

Abb. 1: Grundidee des Requirements-Engineering

Abb. 2: Anforderungs-Breakdown im Überblick

Abb. 3: Rekursive Zerlegung von Anforderungen

Abb. 4: Verwaltete Informationsklassen im Überblick

Abb. 5: Zusammenhang der Anforderungsarten

Literatur & Links

[Hru00] P. Hruschka, Von informellen Wünschen und formalisierten Anforderungen, in: OBJEKTspektrum 2/00

[Oes01]B. Oestereich, P. Hruschka, M. Reinhold, N. Josuttis, H. Kocher, H. Krasemann, Erfolgreich mit Objektorientierung: Vorgehensmodelle und Managementpraktiken für die objektorientierte Softwareentwicklung, Oldenbourg-Verlag, 2001

[Rei97] M. Reinhold, Die UML und das standardisierte Prozessmodell V-Modell'97: Warum reicht eine Modellierungssprache alleine nicht aus?, in: OBJEKTspektrum 5/97

[Rei00] M. Reinhold, Rational Unified Process 2000 und V-Modell'97: Synergie oder Widerspruch, in: OBJEKTspektrum 3/00 (zum Herunterladen unter www.cocoo.de verfügbar)

[Rob01] S. Robertson, Techniken für die Anforderungsanalyse, in: OBJEKTspektrum 1/01

[Rup00] C. Rupp, Requirements-Engineering — der Einsatz einer natürlichsprachlichen Methode bei der Ermittlung und Qualitätsprüfung von Anforderungen, in: OBJEKTspektrum 2/00

[Rup01] C. Rupp, J. Dallner, Mustergültige Anforderungen, in: OBJEKTspektrum 3/01

[VM97] Das V-Modell '97 (siehe www.cocoo.de)