Zielsicher zur bestmöglichen Software

Die Voraussetzungen für eine stringente, durchdachte und kosteneffiziente Software-Entwicklung im Überblick

Hinter dem Begriff „Requirements Management" verbirgt sich eine Aufgabe, ohne die kein Projekt durchgeführt werden kann: Die Spezifikation und Verwaltung der Anforderungen, also der gewünschten Funktionalität und der Eigenschaften der zu entwickelnden Software. Requirements Management ist von entscheidender Bedeutung für das Gelingen des Projekts sowie für die spätere Systemqualität und -funktonalität. Allerdings sind für perfektes Requirement Management sowohl eine sinnvolle Vorgehensweise als auch die Kenntnis diverser Fallstricken nötig, damit aus der zu entwickelten Software letztendlich kein zeit- und kostenintensives „Medusenhaupt" wird.

■ Markus Einsle

Requirements Management ist die Basis für eine erfolgreiche Software-Entwicklung – allerdings gilt es, eine Vielzahl von Anforderungen zu beachten, um die Ziele für die Entwicklung frühzeitig und eindeutig festzulegen und um aufwändige und kostenintensive Änderungen in einer fortgeschrittenen Entwicklungsphase zu vermeiden. Gleichzeitig müssen aber notwendige Änderungen, die eine Konstante in jedem Entwicklungsprozess darstellen, spezifikationsseitig einfach und effizient durchführbar sein, um das Projekt insgesamt innerhalb des vorgesehenen Zeit- und Kostenrahmens abwickeln zu können. Betrachtet man statistisch die reale Erfolgsquote von Softwareprojekten, ergibt sich ein ernüchterndes Bild: Ende 2004 wurden noch immer weniger als ein Drittel aller Projekte erfolgreich abgeschlossen. Beinahe jedes fünfte Projekt scheitert vollständig, der Rest ist von Terminverzug, Mehrkosten und beschnittenem Funktionsumfang geprägt [1]. Ein wesentlicher Anteil davon ist auf Unzulänglichkeiten während der Analysephase zurückzuführen.

In der Praxis weisen Anforderungsspezifikationen in der Gestalt konventioneller Lasten- und Pflichtenhefte trotz enormen Erstellungsaufwands oft erhebliche Mängel auf. Das Spektrum reicht von fehlenden Informationen über implizite oder unpräzise Anforderun- >

Markus Einsle
ist Senior-Consultant für Software-Engineering bei Mixed Mode
T +49/89/89868-200
markus.einsle@mixed-mode.de

	A	B	C	D	E	F	G	H	I	J
1	SMASK:			x	x					
2	Req-ID	Format	Typ	Anforderungstext	Kommentar/ Begründung	Prüfmethode/ Kriterium	Quelle	Prio	Bezuge	
3	SW_REQ_001	0	USR	User Requirements und Constraints						
4	SW_REQ_002		USR	Die Daten der Endbenutzer müssen gegen Zugriff durch Dritte geschützt sein	Kein Zugriff für unberechtigte Personen	Akzeptanztest	Endbenutzer	1	SW_REQ_001	
5	SW_REQ_003		USR	Das Ausgabe-Datenformat des Systems soll kompatibel zum Produkt ABCD sein	Interoperabilität mit Fremdprodukten	Review	Vertrieb	2	SW_REQ_001	
6	SW_REQ_004		CON	Internationale Kunden müssen in der Lage sein, mit dem System zu arbeiten		Akzeptanztest	Vertrieb	1	SW_REQ_001	
7										
8	SW_REQ_100	0	SYS	System Requirements						
9	SW_REQ_101	0	SYS	Authentisierung					SW_REQ_002	
10	SW_REQ_103		SYS	Das System fordert den Benutzer zur Eingabe von Benutzername und Passwort auf			Unit-Test		SW_REQ_101	SW_REQ_004
11	SW_REQ_104		SYS	Das System sperrt das Benutzerkonto, wenn der Benutzer drei mal hintereinander ein falsches Passwort angibt			Unit-Test		SW_REQ_101	SW_REQ_103
12	SW_REQ_102		SYS	Das System identifiziert den Benutzer anhand eines Fingerabdrucks	Alternativ zu SW_REQ_103		Unit-Test		SW_REQ_101	
13	SW_REQ_110	0	SYS	Datenformat					SW_REQ_003	
14	SW_REQ_111		SYS	Das System speichert die Ausgabedaten im Format OpenDocument	file://Standards/OpenDocument				SW_REQ_110	

Abb. 1: Tabellarische Requirements-Notation

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Analysieren Automatik	Alle Suspects löschen	SW_REQ_001	SW_REQ_002	SW_REQ_003	SW_REQ_004	SW_REQ_100	SW_REQ_101	SW_REQ_103	SW_REQ_104	SW_REQ_102	SW_REQ_110	SW_REQ_111
2	SW_REQ_001	User Requirements und Constraints		↗	↗	↗							
3	SW_REQ_002	Die Daten der Endbenutzer müssen gegen Z...	↗					↗					
4	SW_REQ_003	Das Ausgabe-Datenformat des Systems soll...	↗									↗	
5	SW_REQ_004	Internationale Kunden müssen in der Lage...	↗						↗				
6	SW_REQ_100	System Requirements											
7	SW_REQ_101	Authentisierung		↗					↗	↗	↗		
8	SW_REQ_103	Das System fordert den Benutzer zur Eing...				↗		↗		↗			
9	SW_REQ_104	Das System sperrt das Benutzerkonto, wen...						↗	↗				
10	SW_REQ_102	Das System identifiziert den Benutzer an...						↗					
11	SW_REQ_110	Datenformat			↗								↗
12	SW_REQ_111	Das System speichert die Ausgabedaten im...										↗	

Abb. 2: Suspect Links in der Traceability-Matrix

gen, die vom Auftragnehmer anders interpretiert werden als vom Auftraggeber gedacht, über nicht erfüllbare Anforderungen bis hin zu unentdeckten Widersprüchen in ein und demselben Dokument. Doch damit nicht genug: während der Entwicklungsphase entsteht aufgrund der zwischenzeitlich gewonnenen Erfahrung, aufgetretenen Problemen und Veränderung der äußeren Umstände fast in jedem Projekt unweigerlich Änderungsbedarf. Der Aufwand, alle bis dahin existierenden Projektartefakte zu überarbeiten, steigt dabei überproportional mit dem Projektfortschritt.

Alternative: tabellarische Darstellung

Um nicht fortwährend in dieselbe Sackgasse zu geraten, sollten alternative Notationsmethoden in Erwägung gezogen werden. Denn mit der Wahl der richtigen Notationsmethode steht und fällt die Machbarkeit einer durchgängig konsistenten Spezifikation/Dokumentation über die gesamte Projektlaufzeit. Bewährt hat sich hier anstelle von Prosa eine tabellarische Form. Jede Einzelanforderung wird dabei separat und atomar festgehalten. Sie besteht aus einem innerhalb des Projektes eindeutigen Identifier, dem eigentlichen Anforderungstext, sowie zusätzlichen Attributfeldern (siehe Abbildung 1).

Identifiers dienen zur eindeutigen Kennzeichnung von Requirements, die sich auf diese Weise beliebig referenzieren lassen und so eine durchgängige Verfolgbarkeit gewährleisten – falls erforderlich, bis in die Implementierungsebene und zurück. Aufgrund der fixen Zuordnung der Identifiers ist es möglich, die Requirements beliebig im Dokument zu verschieben, ohne die Konsistenz zu gefährden. Voraussetzung: die Identifiers sind projektweit eindeutig und werden im Projektverlauf weder verändert noch gelöscht.

Da die menschliche Sprache inhärent uneindeutig ist, soll jeder Anforderungstext kurz, prägnant und präzise formuliert werden. Unerlässlich ist ein projektweites Glossar, in dem Fachausdrücke genau definiert werden. Gerade wenn Experten aus mehreren Disziplinen beteiligt sind, ist bei Ausdrücken, die in mehreren Fachjargons abweichende Bedeutung besitzen, Vorsicht geboten. Selbst unter dem scheinbar trivialen Begriff „Funktion" verstehen beispielsweise Mathematiker, Softwareingenieure und Maschinenbauer unterschiedlichste Dinge!

Eines der wichtigsten Attribute ist das Prüfkriterium. Es sichert die Verifizierbarkeit der Anforderung. Grundsätzlich gilt: nur eine prüfbare Anforderung ist eine gültige Anforderung. Dabei ist es zunächst weder notwendig, noch möglich, einen detaillierten Testfall zu hinterlegen. Es genügt im ersten Schritt die Angabe, auf welche Art und Weise eine Prüfung durchführbar ist, z.B. mit Hilfe eines Akzeptanztests, Unit-Tests oder per Code-Review. Nicht objektiv prüfbar und somit ungültig ist beispielsweise die stark subjektive Anforderung, die Bedienoberfläche „anwenderfreundlich" zu gestalten.

Der Umgang mi Änderungen

Kommt es nun zu einer Änderung an einem Requirement, sind jeweils gleichzeitig die über ihre Identifiers verknüpften Requirements zu überprüfen. Dadurch kann zwar eine wahre Änderungslawine ausgelöst werden. Was auf den ersten Blick abschrecken mag, macht jedoch im Gegensatz zu konventionellen Spezifikationsmethoden den gesamten und notwendigen Änderungsbedarf transparent. Der bei konventionellen Textdokumenten entstehende Aufwand, große Teile der Spezifikation neu zu formulieren, strukturell umzugestalten und zu formatieren, entfällt dagegen.

Spezifizierung

Nachdem solchermaßen technisch günstige Rahmenbedingungen geschaffen wurden, besteht die Kunst des Requirements Managements nun darin, die zum jeweiligen Entwicklungszeitpunkt angemessene Granularität der Spezifikation zu erreichen. Die wesentlichen Ziele müssen klar und unmissverständlich wiedergegeben werden, ohne jedoch die Realisierung vorweg zu nehmen. Hilfreich ist es dabei, die Requirements auf mehreren hierarchischen Ebenen anzuordnen.

Die oberste Ebene beschreibt ausschließlich die User-Requirements. Das sind die Anforderungen realer und abstrakter Personen, die Interessen am Produkt geltend machen, darunter Endbenutzer, Produktion, Vertrieb und Gesetzgeber. Aber auch technische Systeme, die Schnittstellen zum Produkt aufweisen, lassen sich als „Users" betrachten. User-Requirements beschreiben die Erwartungen des Users an das System, und stellen die Grundlage für den Akzeptanztest dar.

System-Requirements beschreiben dagegen, was das System leisten muss, um den User-Requirements zu genügen. Gerade im Softwarebereich existiert meist eine Vielzahl alternativer System-Requirements, die diesen Zweck erfüllen. Hier sind Know-How und Kreativität des Lieferanten, sowie eine enge Abstimmung mit dem Kunden gefragt. Eine fundierte Entscheidung über die zu realisierenden System-Requirements, die sich durchaus auch auf das Software-Design auswirken wird, muss durch beide Parteien gefällt werden. System-Requirements stellen die Grundlage für den Software-Entwickler dar, der diese Requirements im Quellcode realisiert.

In einer korrekten Requirements-Spezifikation bestehen stets m-zu-n Beziehungen zwischen beiden Requirements-Typen, die sich durch Verknüpfungen ausdrücken lassen. In der Regel wird ein System in Subsysteme aufgeteilt, wobei für jedes Subsystem erneut User- und

(Sub-)System-Requirements aufgestellt werden können. Als „User“ wird auf Subsystemebene in der Regel nicht mehr ein realer Anwender aufgefasst, sondern das darüber liegende System und seine Entwickler.

Neben User- und System-Requirements lässt sich der Typus der Constraints unterscheiden. Constraints beschreiben fixe Rahmenbedingungen, z.B. ein vorgegebenes Betriebssystem, notwendiges Zeitverhalten oder Schnittstellen zu bereits existierenden Hard- und Softwaresystemen. Constraints wirken querschnittlich und unabhängig von den Requirements-Hierarchieebenen auf das gesamte System.

Embedded System Software

Im Kontext von Embedded-Software ergibt sich eine besondere Situation. Verkauft wird hier letztendlich ein Gesamtprodukt. Dass dieses aus Software, Hardware und ggf. mechanischen Bestandteilen besteht, ist aus Kundensicht von untergeordneter Bedeutung. Die Anforderungen des Kunden konzentrieren sich folglich auf das Gesamtsystem als Black Box und seine äußeren logischen, elektrischen und ggf. mechanischen Schnittstellen, sowie Constraints bzgl. Performance, Lebensdauer und Zuverlässigkeit.

Die Klammer um das gesamte Embedded-System bilden – analog zur reinen Software-Entwicklung – die User-Requirents. Bevor daraus jedoch System-Requirements abgeleitet werden können, müssen die Subsysteme aus den Bereichen Software, Hardware und Mechanik etc. grob abgegrenzt werden. Beim Finden der System-Requirements ist abzuwägen, welchem dieser Subsysteme die tragende Rolle für das jeweilige System-Requirement zugesprochen wird. Z.B. könnte die Begrenzung der Bewegungsbahn eines Stellmotors sowohl vollständig in Software, durch elektrische Abschaltung mit Hilfe eines Endlagenschalters, oder durch mechanische Entkopplung gelöst werden. Die Freiheitsgrade bei der Ermittlung der System-Requirements sind daher deutlich größer als bei reinen Software-Systemen – was das Projekt jedoch auch deutlich komplexer macht und eine straffe Koordination der Subsysteme erfordert. Essentiell sind daher subsystemübergreifende Verknüpfungen, die sicherstellen, dass der von der Software angesteuerte Stellmotor auch tatsächlich verbaut wird, eine elektrische Treiberstufe auf der Platine vorhanden ist, und Änderungen sich übergreifend fortpflanzen.

Ein wesentlicher Unterschied besteht bei Embedded-Systemen in der Art der Benutzerschnittstelle. Applikationssoftware besitzt in der Regel eine direkte Schnittstelle zum (menschlichen) Anwender, an der ein großer Anteil der Requirements festgemacht und weiter spezifiziert werden kann. Während hier zum Auffinden der Requirements eine Kombination von Use Cases und Click-Dummies – d.h. prototypisch realisierte Eingabemasken mit minimaler bzw. fehlender Funktionalität – hervorragende Dienste leisten, ist bei Embedded-Applikationen die Schnittstelle zum Endanwender oft sehr schlank ausgebildet, z.B. in Form weniger Bedientasten. Teile des Systems können sich dennoch sehr komplex gestalten, da eine Vielzahl von Eingangsgrößen wie Messwerte, vor allem aber innere Systemzustände in das Systemverhalten mit eingehen. Die Anzahl der Möglichkeiten wachsen exponentiell mit der Mächtigkeit des Eingangsvektors, so dass für ein scheinbar einfaches Teilsystem leicht mehrere hundert Fälle zu berücksichtigen sind, die bei der Entwicklung spezifiziert werden müssen. Es ist hierbei nicht immer zweckmäßig, jeden möglichen Pfad als textuelles Requirement zu formulieren. In solchen Fällen ist der punktuelle und gezielte Einsatz z.B. von Zustandsübergangs-Diagrammen, Sequenzdiagrammen, Wahrheitstabellen etc. sinnvoller und besser für eine genaue Spezifikation geeignet. In der Requirements-Spezifikation werden lediglich die grundlegenden Regeln textuell hinterlegt, und mit Hilfe eines Verweises das jeweilige Drittdokument in die Spezifikation integriert.

Fazit

Requirements Management ist mit Sicherheit nicht das alleinige Allheilmittel, um die Probleme bei der Entwicklung der stetig komplexer werdenden Software vollständig in den Griff zu bekommen. Zusammen mit Kernpraktiken wie durchgängigem Änderungsmanagement, Konfigurationsmanagement und Software-Tests stellt Requirements Management jedoch einen wichtigen Baustein auf dem Weg zu qualitativ hochwertiger Software dar. Nur eine äußerst durchdachte, stringente Vorgehensweise, angewandt von einem erfahrenen, kompetenten Team, stellt die Entwicklung einer einwandfrei funktionierenden Software im vorgegebenen Kostenrahmen sicher. Es empfiehlt sich daher immer, frühzeitig auf das Fachwissen von erfahrenen Experten zurückzugreifen, um teure Fehlentwicklungen von Anfang an zu vermeiden. ■

Literatur

[1] The Standish Group International, Inc.: 2005 third quarter research report http://standishgroup.com

Weiterführende Infos auf www.EuE24.net

more @ click EE125301