Bottom-up Use-Cases

Peter Hruschka

Seit vielen Jahren haben sich Use-Cases [Jac92] bewährt, um komplexe Systeme aus Anwendersicht in Prozesse aufzuteilen, die anschließend getrennt voneinander im Detail analysiert werden können. Die Vorgehensweise, Systemanalyse mit Use-Cases zu beginnen, diese anschließend in Schrittfolgen aufzuteilen und somit zu einer vollständigen Spezifikation zu kommen, ist in vielen Lehrbüchern und Vorgehensmodellen beschrieben. In diesem Beitrag wollen wir Use-Cases aus einer anderen Sicht heraus beleuchten: als Mittel, um eine große Menge zunächst unkoordiniert geäußerter Anforderungen, die unter Umständen von einer Vielzahl von Stakeholdern kommen, zu systematisieren und in ein beherrschbares Gesamtdokument zu überführen.

Ivar Jacobson hat mit seiner Veröffentlichung seines Use-Case-getriebenen Ansatzes im Jahre 1992 einen Standard gesetzt, der einen weltweiten Siegeszug angetreten hat. Die Gründe für diesen Siegeszug liegen meines Erachtens nach in zwei Eigenschaften der Use-Cases: erstens sind Use-Cases eine Zerlegung eines komplexen Systems aus neutraler Außensicht - aus der Sicht von Akteuren, die Leistung von dem System haben wollen. Damit ist es nicht notwendig, sich frühzeitig auf interne Strukturen im System festzulegen. Und zweitens hat Ivar Jacobson vielen Anwendern ihre natürliche Sprache zurückgegeben, indem er vorgeschlagen hat, Use-Case-Spezifikationen zunächst umgangssprachlich zu formulieren. Damit wurden die Spezifikationen wieder für IT-Laien verständlich, lesbar und kommentierbar.

Fast jedes neu gestartete Projekt nutzt Use-Case Analyse, um das geplante Produkt oder System zunächst unter Nutzung der beiden oben genannten Vorzüge zu starten. Obwohl in der Praxis auf auch Use-Case-Modelle erstellt werden, die mit den Originalideen von Ivar Jacobson außer dem Namen nichts gemein haben, hat sich die Denkart doch in vielen Projekten etabliert. Üblicherweise arbeiten solche Projekte mit einem einzigen Analytiker oder einer kleinen Gruppe von Requirments-Ingenieuren, die diese systematische Aufteilung vornehmen, bevor sie dann pro Use-Case aus dem Kunden alle Details herauslocken, die für eine vollständige Spezifikation notwendig sind. In diesem Beitrag wollen wir von einer anderen, aber auch sehr geläufigen Situation ausgehen: von einem Projekt, im dem eine Vielzahl von Personen Wünsche und Forderungen an den nächsten Release stellt. Üblicherweise spricht jeder Projektbeteiligte oder Projektbetroffene nur über die Dinge, die ihm oder ihr besonders am Herzen liegen. Auf diese Art entstehen mehr oder weniger umfängliche „Feature-Listen“ oder große Sammlungen von Einzelanforderungen, die zunächst keine inhärente Ordnung haben.

Featuritis

Diese Vielzahl von geforderten Features zu verwalten und zu versionieren, macht angesichts der heute verfügbaren Requirements-Management-Tools technisch gesehen

keine großen Schwierigkeiten. Die Schwierigkeiten im Projekt sind eher in der menschlichen Beherrschbarkeit von solchen Featuremengen zu suchen. Ein gewünschtes Feature für ein Produkt kann alles Mögliche sein: eine kleine, neue Funktionalität, die das Produkt können soll, eine Forderung nach einem bestimmten Erscheinungsbild an der Oberfläche, eine Performanzanforderung, ein großes neues Subsystem mit umfangreicher Funktionalität, die bisher vom Produkt noch nicht geboten wird, und vieles mehr. Features, die von vielen Stakeholdern kommen, sind also ein bunte Mischung aus groben und feinen funktionalen Anforderungen, nichtfunktionalen Anforderungen und Randbedingungen für das zu entwickelnde Produkt. Würde man versuchen, diesen Sachverhalt grafisch auszudrücken, so entstünde etwas, wie in Abbildung 1 dargestellt.

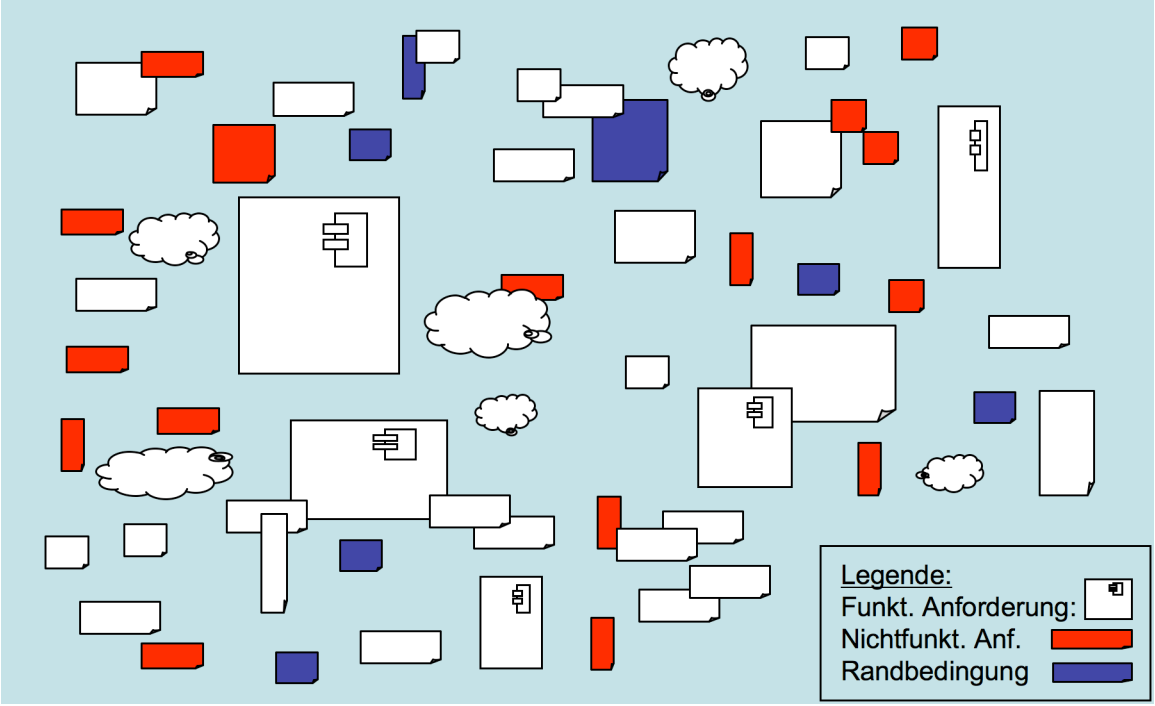


Abb. 1: Unterschiedliche Arten von Features auf unterschiedlichem Abstraktionsniveau

Einen weniger abstrakten Ausschnitt über derartig gestaltete Anforderungen finden Sie in der Box mit dem Beispiel (Abbildung 2). Bitte beachten Sie, dass die Wünsche in der Form von irgendwelchen Projektbeteiligten so geäußert wurden und nicht unbedingt dam State-of-the-Art guter Requirementsformulierung genügen.

Neue Wünsche an das AQUARIUS-System:

.....

17. Das Aquarius-System soll die Startnummern für Wettkämpfe nach dem Prinzip „First-Come – First-Serve“ vergeben.
18. Die neue Version soll auch Anmeldung per SMS zulassen.
19. Wir wollen einen PC für die Lösung einsetzen, der für unter 1000 Euro erhältlich ist.
20. Wenn ein Kind am „übernächsten Sonntag“ mitschwimmen möchte, so soll das

System den Wettkampf auch über solch relative Angaben identifizieren können.
21. Aquarius soll Siegerurkunden drucken, wenn nicht genügend Sachpreise vorhanden sind.
22. Bei der Bewertung ist der Mittelwert der Kampfrichterwertungen zu verwenden, nachdem zunächst die höchste und tiefste Punktzahl gestrichen wurden
23. Im Saisonplan ist die Liste der geplanten Figuren für jeden Wettkampf zu veröffentlichen, um den Kindern das rechtzeitige Trainieren der Figuren zu ermöglichen.
24. Das System soll prüfen, dass Kinder, die sich zu einem Wettkampf anmelden, bei einem Team gemeldet sind.
25. Falls ein Kind Figuren vorführen möchte, die nicht für den Wettkampf vorgesehen sind, soll Aquarius Alternativen vorschlagen, bzw. einen Wettkampf, wo die Figuren geplant sind.
26. Der Schwierigkeitsfaktor der gezeigten Figur soll bei der Berechnung der Endpunkt berücksichtigt werden.
27. Die Plausibilitätsprüfungen eines Kindes im Zuge der Anmeldung zu einem Wettkampf dürfen nicht länger als 3 Sekunden dauern.
28. .....

Abb. 2: Eine Sammlung von Features

Derartige Featurelisten haben für die Gestaltung des weiteren Projektablaufs unangenehme Eigenschaften:

- Sie mischen Wichtiges mit Unwichtigem, Details mit großen Funktionen
- Sie sind wegen der Menge (für menschliche Leser) oft schlecht beherrschbar, wenn es um Prioritätsentscheidungen, bzw. um die Auswahl von Funktionen für die nächste Version geht
- Sie verschleiern die Abhängigkeiten untereinander

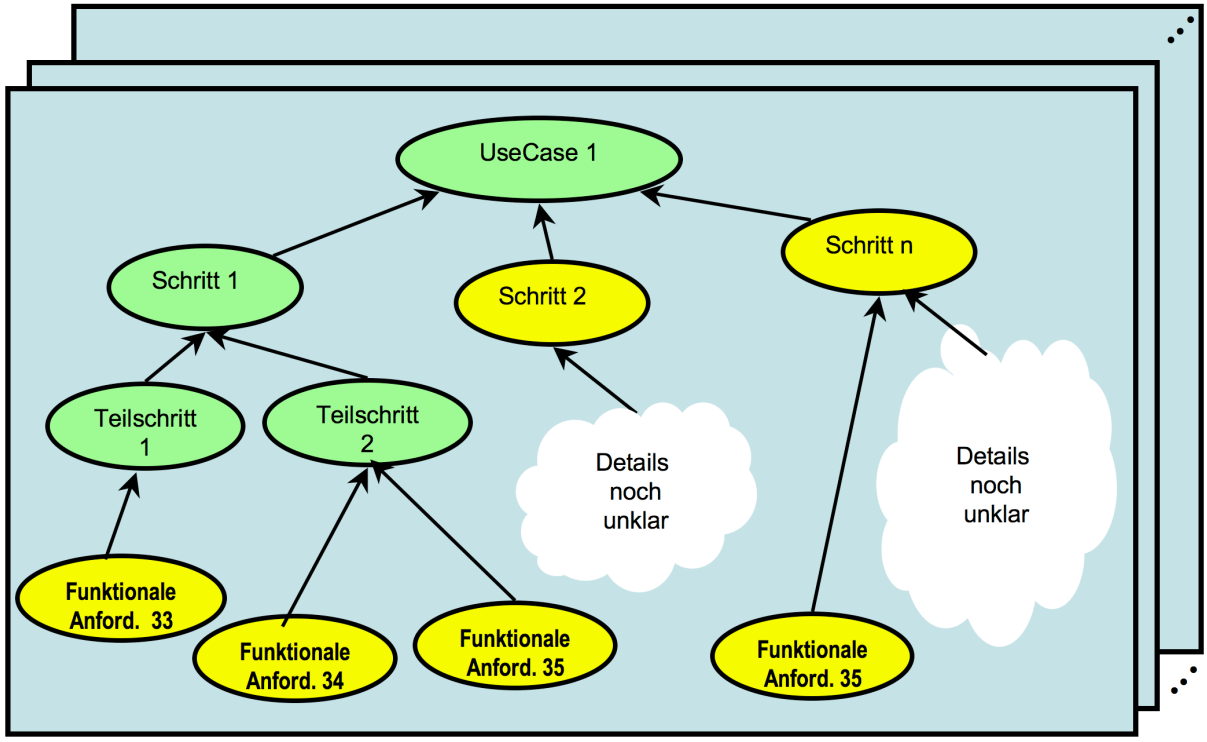
Kurz gesagt: sie erschweren gezielte Entscheidungen im Projekt.

Bottom-Up Sortierung der funktionalen Anforderungen

Mein Vorschlag ist es, eine derartige Sammlung von Features nicht für sich bestehen zu lassen, und zu versuchen, sie zu beherrschen, sondern nachträglich in diese Sammlung Ordnung zu bringen, die dem heutigen Stand der Kunst im Requirements Engineering entspricht. Diese Ordnung stellen Sie bottom-up her. Sie versuchen, die Menge in den Griff zu bekommen, indem Sie das Wissen ausnützen, dass jede (funktionale) Anforderung zu irgendeinem Schritt irgendeines Use-Cases gehört. Dahinter steckt folgende Annahme:

- Alle Funktionen, die das System ausführen soll, sollen als Use-Case modelliert sein. Oder anders ausgedrückt: die Menge aller Use-Cases zusammengenommen entspricht der vollständigen Systemfunktionalität
- Use-Cases lassen sich in Schritte zerlegen, die den normalen Ablauf und alle Sonderfälle widerspiegeln. Sollten die Schritte noch zu grob sein, dann kann man jeden Schritt weiter in Teilschritte zerlegen. (In der UML nutzen Sie dafür Aktivitätsdiagramme.)

Sie versuchen also, für jede genannte funktionale Anforderung in der Feature-Liste einen passenden Use-Case zu definieren. Damit erhalten Sie z.B. statt 487 Einzelanforderungen eine Gliederung in 18 Use-Cases mit im Schnitt 26 Anforderungen. Diese können Sie jetzt innerhalb des Use-Cases noch in eine vernünftige Ablaufreihenfolge bringen (Vgl. Abbildung 3).



Gelb: von Kunden gefordert
Grün: vom Analytiker als Zusammenhang entdeckt
Weiß: noch keine Angaben, muss noch erfragt werden

Abb. 3: Funktionale Anforderungen gemäß Use-Cases und Teilaktivitäten sortieren

Betrachten wir das an einigen Beispielen aus der oben genannten Feature-Liste:

> 17. Das Aquarius-System soll die Startnummern für Wettkämpfe nach dem Prinzip „First-Come – First-Serve“ vergeben.

Der Analytiker versucht den externen Auslöser für die Vergabe von Startnummern zu finden. Der Kunden klärt ihn auf, dass Startnummern direkt bei der Anmeldung eines Kindes zum Wettkampf vergeben werden. Somit haben wir einen Use-Case des Systems entdeckt und können diese Anforderungen dem Use-Case „Wettkampfanmeldung“ zuordnen. Wir kennen jetzt auch den Akteur: ein Kind. Durchforsten wir die Features weiter, so finden wir heraus, dass auch Nummern 20, 24 und 25 zur Anmeldung gehören. Wir ordnen sie ebenfalls diesem Use-Case zu. Nach einigen weiteren klärenden Gesprächen mit den Auftraggeber finden wir heraus, dass Feature 21 zum Use-Case „Wettkampfauswertung“ gehört, während die Forderung 22

und 26 zum „Versuch bewerten“ gehören. Daraus ergibt sich dann ein Zwischenstand, wie in Abbildung 4 dargestellt.

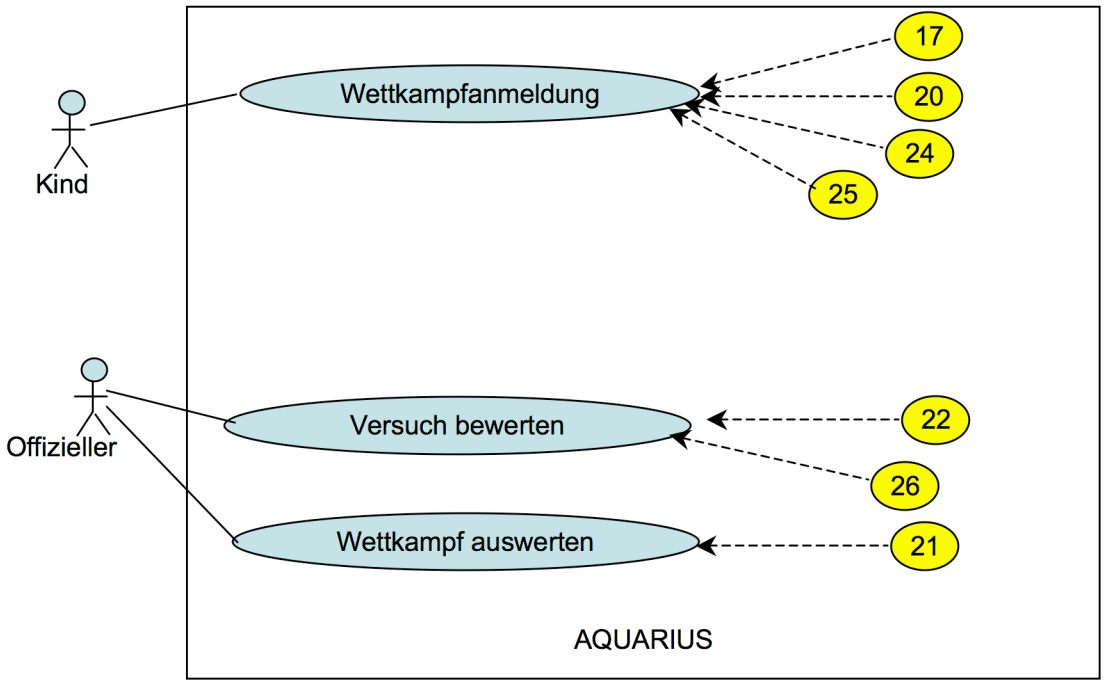


Abb. 4: Features sortiert nach Use-Cases

Wir sind uns natürlich bewusst, dass wir mit der Zuordnung nicht unbedingt alle Use-Cases des Systems gefunden haben, aber das war ja auch nicht das vorrangige Ziel. Es ging uns vielmehr darum, die genannten funktionalen Anforderungen mengenmäßig besser beherrschbar zu machen – und Use-Cases sind ein anerkanntes Bündelungsprinzip dafür. Für die wenigen Anforderungen unseres Beispiels ist diese Anordnungsübung natürlich einfach, aber denken Sie an reale Featurelisten mit hundert oder mehr Anforderungen. Dann ist die Wirkung dieser Gliederung deutlicher sichtbar.

Als nächstes Zwischenziel können wir jetzt versuchen, Use-Cases mit vielen zugeordneten Anforderungen noch weiter zu strukturieren, indem wir die Anforderungen in eine zeitliche Reihenfolge bringen, also jede Anforderung einem Use-Case-Schritt zuordnen. Für unser kleines Beispiel finden wir heraus, dass Nr. 17 (Startnummern vergeben) am Schluss der Anmeldung passiert, dass 20 und 25 im Zuge der Wettkampfüberprüfung gemacht werden und 24 bei der Prüfung der Kinder. Außerdem erläutert unser Kunde, dass die Wettkampfprüfung und die Kinderüberprüfung unabhängig voneinander in beliebiger Reihenfolge gemacht werden können. Nach Einarbeitung dieser Informationen ergibt sich der in Abbildung 5 dargestellte Zwischenzustand.

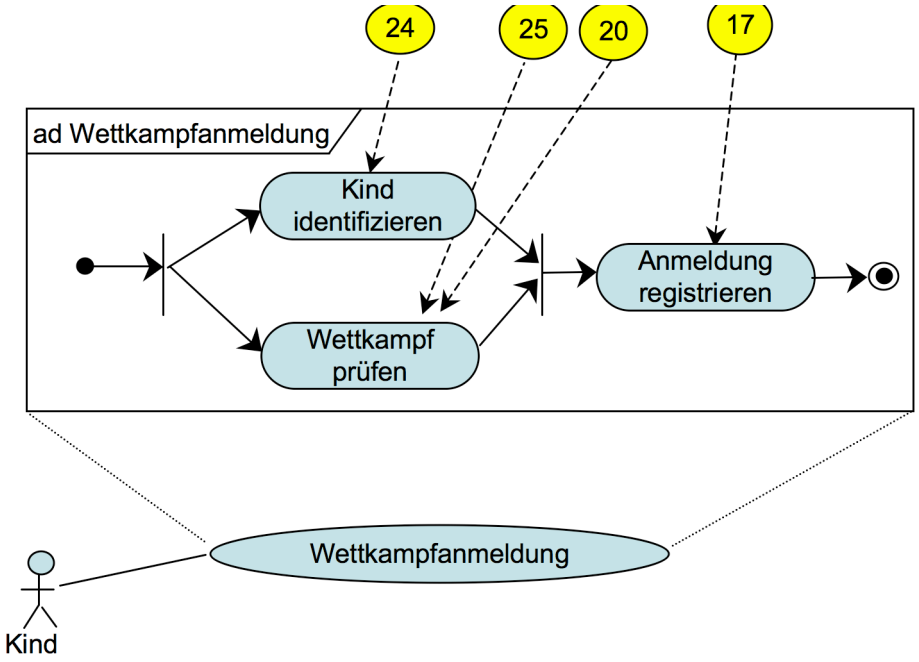


Abb. 5: geordnete Use-Case-Schritte

Falls gewünscht lassen sich jetzt auch viel gezielter Fragen zu dem Gesamtablauf stellen. Z.B. Was muss sonst noch alles bezüglich der Identifikation der Kinder geprüft werden? Worauf kommt es bei der Wettkampfprüfung an? Was außer der Startnummer wir bei der Registrierung der Anmeldungen getan. Auf diese Wiese kommen Sie zu einer Vervollständigung Ihres Verständnisses der Anforderungen, die anhand der Featureliste nicht möglich gewesen wäre. Die nachträgliche Einbettung jeder funktionalen Anforderung in einen Use-Case hilft also nicht nur, die Menge der Kundenwünsche zu ordnen und beherrschen, sondern auch dabei, Lücken und Widersprüche aufzudecken.

Was passiert mit den restlichen Anforderungen?

Die Feature-Liste enthält neben den funktionalen Anforderungen, die wir als Fragmente von Use-Case anordnen können, aber auch nicht-funktionale Anforderungen und Randbedingungen. In Abbildung 2 fallen die Anforderungen 18, 19 und 27 in diese Kategorie. Was machen wir mit diesen bei unserer Bottom-up-Strategie? Zunächst können wir sie kategorisieren wie in VOLERE [Rob06] oder ARTE [Hru02] vorgeschlagen. Danach lassen sich viele davon auch den bottom-up-organisierten Schritten und Use-Cases zuordnen. 18 gehört zur Kategorie „geforderte Technologien“, 19 ist eine Kostenanforderungen und 27 eine Performanzanforderung. Zwei davon (18 und 27) betreffen den Use-Case, den wir oben genauer betrachtet haben. 18 betrifft den ganzen Use-Case, wird also auf dieser Ebene zugeordnet; 27 betrifft den Schritt des Identifizierens von Kindern. Danach ergibt sich folgendes Gesamtbild.

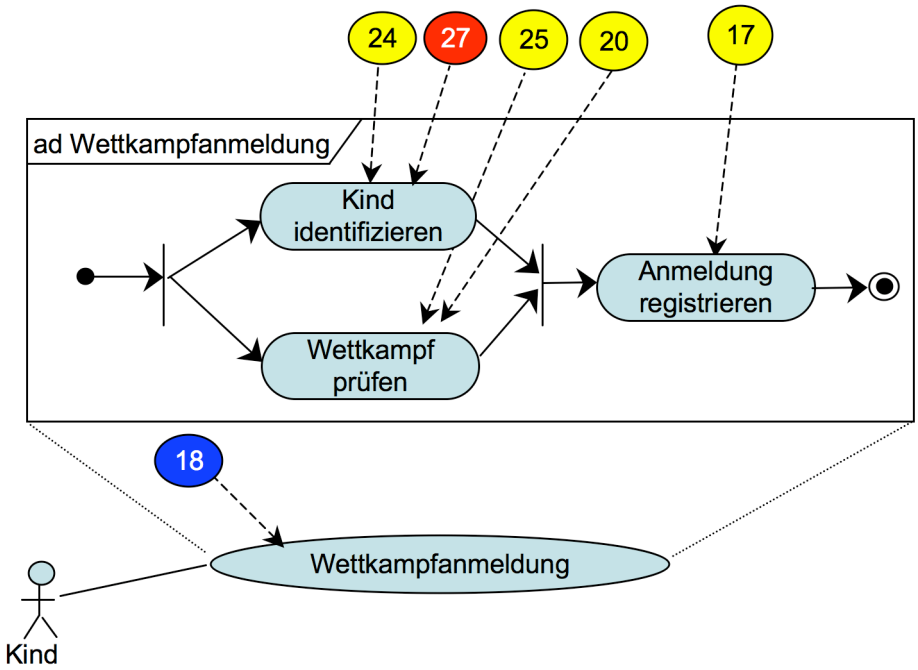


Abb. 6: Zuordnung von nicht-funktionalen Anforderungen und Randbedingungen

In der Kürze dieses Artikels haben wir außer Acht gelassen, dass moderne Requirements-Methoden und die UML noch mehr an Strukturierung von großen Listen von Anforderungen erlauben. So können Anforderungen nicht nur durch Use-Case-Modelle geordnet werden, sondern auch Begriffsdefinitionen, fachlichen Klassenmodellen, oder auch Zustandsautomaten zugeordnet werden. Ein ausführlicheres Beispiel über die Gliederung von Anforderungen in einem realen industriellen Projekt findet sich in [Flex05].

Fazit

Use-Cases werden heute in der Systemanalyse gerne eingesetzt und von Anwendern gut verstanden. Wir haben sie in diesem Beitrag zweckentfremdet: als nachträgliches Kriterium zur Sortierung und Anordnung einer Vielzahl von großen und kleinen Produktfeatures, Wünschen und Randbedingungen. Für die Projektpraxis empfehlen wir Ihnen, selbst bei einer großen Anzahl von Stakeholdern und einer Menge von unkoordiniert eingehenden Anforderungen und Änderungswünschen diese erst gar nicht unstrukturiert zu erfassen, sondern so schnell wie möglich die Ordnung in Form von Use-Cases und deren Schritten herzustellen. Sie haben damit ein Mittel in der Hand, um nicht nur Überblick zu schaffen, sondern auch gezielt Lücken und Widersprüche in den Anforderungen aufzudecken.

Literatur:

[Flex05] http://www.flexray.com/specification_request_v21.php : In dem Publication-Bereich finden Sie auch die Requirements-Spezifikation dieses Bussystems der Automobilindustrie.

[Hru02] Peter Hruschka, Chris Rupp: Agile Software-Entwicklung für Embedded Real-Time Systems mit UML. Carl-Hanser-Verlag, 2003

[Jac92] Ivar Jacobson: Object Oriented Software Engineering - A Use Case Driven Approach, Addison Wesley, 1992 .

[Rob06] James Robertson, Suzanne Robertson: Mastering the Requirements Process, Second Edition, Addison Wesley, 2006.