Planguage – Spezifikation nichtfunktionaler Anforderungen

Torsten Emmanuel

Einleitung

Nichtfunktionale Softwareeigenschaften sind mindestens so wichtig wie die funktionalen; mitunter sind sie sogar wettbewerbsentscheidend. Ihre Spezifikation macht jedoch besondere Schwierigkeiten, sodass sie oft nicht ausreichend und in geeigneter Weise berücksichtigt werden. Dieser Beitrag beschreibt die Ursache und stellt mit Planguage von Tom Gilb [5] eine Methode vor, die helfen soll, das Problem zu lösen.

Nichtfunktionale Anforderungen – was macht sie so besonders?

Typischerweise werden Anforderungen unterteilt in funktionale Anforderungen und Anforderungen, die keine Funktion repräsentieren, und deshalb nichtfunktional genannt werden. Sie stellen einen besonderen Anforderungstyp dar:

- Nichtfunktionale Anforderungen werden häufig erst erfahr-, test- und messbar, wenn das Gesamtsystem steht.
- Nichtfunktionale Anforderungen beeinflussen sich gegenseitig und hängen voneinander ab. Beispielsweise macht ein höheres Maß an Softwaresicherheit zusätzliche Sicherheitsvorkehrungen erforderlich, die wiederum die Antwortzeiten verlängern können.
- Nichtfunktionale Anforderungen sind typischerweise schwieriger zu implementieren und zu gewährleisten.

Das größte Problem jedoch ist, dass nichtfunktionale Anforderungen während der Spezifikation stärker vernachlässigt werden. Aus Zeitmangel oder weil sie zu einem frühen Zeitpunkt schwieriger zu spezifizieren sind, werden sie übersehen oder sogar bewusst nur vage formuliert. Fehler, die in einer frühen Phase der Systementwicklung gemacht und erst in späteren Phasen behoben werden (wenn es dafür eigentlich schon zu spät ist), sorgen aber für einen Summationseffekt, bei dem sich die Fehler fortpflanzen und potenzieren [1, S. 1] und [12, S. 14]. Das bedeutet, dass die Integration nichtfunktionaler Anforderungen zu einem späteren Projektzeitpunkt deutlich zeitintensiver und teurer ist [4, S. 68]. Der unzureichende Umgang mit nichtfunktionalen Anforderungen stellt deshalb aus Projektsicht eines der größten Risiken dar.

Die Frage liegt nahe, ob es nicht geeignete Methoden und Vorgehensweisen gibt, damit umzugehen und die Risiken zu vermeiden. Ein Blick in die Literatur zeigt, dass es keine einheitliche, allgemein akzeptierte Definition für nichtfunktionale Anforderungen gibt, wie es für Funktionen der Fall ist. Je nach Autor wird der Begriff unterschiedlich besetzt, so definiert der Begriff

- Eigenschaften und Einschränkungen [3, 7],
- nur Einschränkungen [8],

DOI 10.1007/s00287-010-0435-5
© Springer-Verlag 2010

Torsten Emmanuel
msg systems ag,
Robert-Bürkle-Str. 1, 85737 Ismaning
E-Mail: torsten.emmanuel@msg-systems.com

*Vorschläge an Prof. Dr. Frank Puppe <puppe@informatik.uni-wuerzburg.de> oder Prof. Dr. Dieter Steinbauer <dieter.steinbauer@schufa.de>

Alle „Aktuellen Schlagwörter" seit 1988 finden Sie unter: www.ai-wuerzburg.de/as

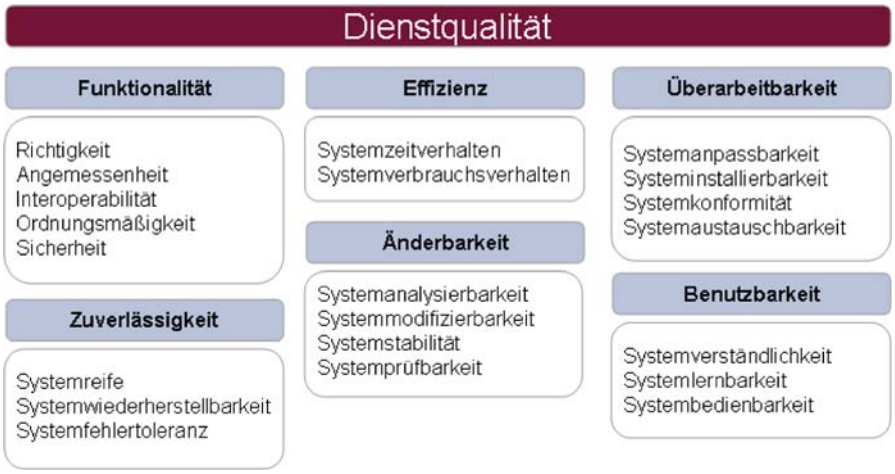

Abb. 1 Dienstqualität nach DIN EN ISO 66272 (in Anlehnung an Rupp et al. [12, S. 17])

– nur Verhaltenseigenschaften [10] und
– Eigenschaften oder Qualitäten [11].

Eine gute Zusammenfassung unterschiedlichster Definitionen findet sich bei M. Glinz [6]. Sie zeigt unter anderem, dass in den Standards IEEE 610.12 (Glossary of software engineering terminology) und IEEE 830-1998 (Recommended Practice for Software Requirements Specifications) der Begriff gar nicht definiert ist.

Auch eine allgemein akzeptierte Sammlung von Attributen oder Kategorien, die nichtfunktionale Anforderungen repräsentieren, gibt es nicht. Je nach Autor werden damit die unterschiedlichsten Aspekte adressiert: Antwortverhalten, Zuverlässigkeit, Benutzbarkeit, Testbarkeit, Wartbarkeit, Kompatibilität, Vorgaben bzgl. Einsatz von Entwicklungswerkzeugen und Bibliotheken, Qualität, rechtlich-vertragliche Belange oder wirtschaftliche und gesetzliche Einschränkungen.

Der Bezugspunkt spielt dabei ebenfalls eine Rolle. Einige nichtfunktionale Anforderungen wirken lokal auf einen Anwendungsfall, andere eher systemglobal als Art Rahmenbedingung, wie z. B. das Look & Feel [11, S. 174].

Wie breit das Spektrum ist, zeigt ein Blick auf die Norm DIN EN ISO 66272, deren Begriff der Dienstqualität von einigen Autoren unter anderem stellvertretend für nichtfunktionale Eigenschaften referenziert wird (Abb. 1).

Planguage, der Ansatz von Tom Gilb

Mit seiner Methode des „Competitive Engineering“ will Tom Gilb dazu beitragen, die kritischen Erfolgsfaktoren zu beherrschen und sicherzustellen, dass Systeme erstellt werden, die im Wettbewerb bestehen. Im Mittelpunkt steht dabei Planguage (ausgesprochen wie englisch „language“ mit der Anfangssilbe „plan“ statt „lan“).

Planguage steht für Planning Language und bezeichnet zugleich eine formelle Spezifizierungssprache und eine Sammlung von Methoden und Prozessen für das System Engineering.

Planguage zählt zu den Spezifikationstechniken, die versuchen, die natürlichsprachliche Spezifikation durch zusätzliche Aspekte zu verfeinern. Für die Spezifikation stehen ein umfangreiches Glossar (von A wie „And“ [als logischer Operator], bis W wie „Wish“ [gewünschter Zielparameter]), ein Set von Parametern (zum Beispiel steht „< – >“ für „fuzzy“ und markiert einen noch nicht ausreichend klar spezifizierten Begriff oder Abschnitt der Spezifikation) und einige Icons zur Vereinfachung bereit. Diese Mittel helfen bei der Strukturierung, mit dem Ziel, eigenständige, klar abgegrenzte Einzelanforderungen hervorzubringen, um diese individuell betrachten und managen zu können. Mithilfe dieser Technik werden innerhalb von Planguage sowohl Anforderungen als auch Designkonzepte und Projektpläne erstellt.

Bevor wir auf die nichtfunktionalen Anforderungen eingehen können, muss zunächst dargestellt werden, wie bei Gilb [5, S. 48] die Einflussfaktoren und Eigenschaften („System-Attribute“) eines Systems im Zusammenhang stehen (Abb. 2).

Jedes System stellt dabei unter einschränkenden Rahmenbedingungen (Conditions), mit bestimm-

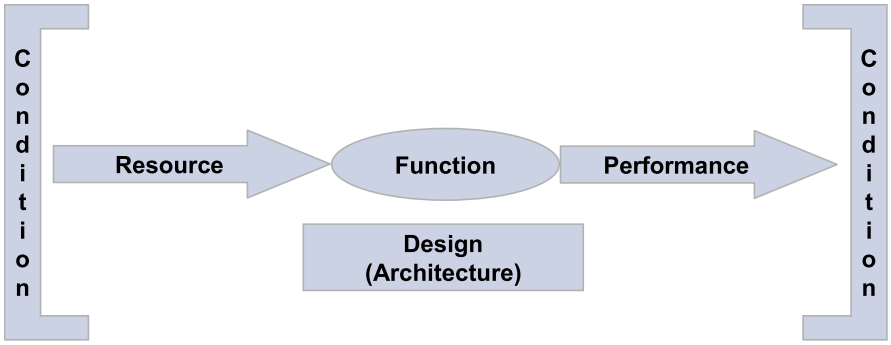

Abb. 2 Zusammenhang der Systemattribute eines Systems (aus Gilb [5, S. 47])

ten gegebenen Mitteln (Resources) und auf Basis spezifischer Designkonzepte (Design/Architecture), Funktionen mit einer individuellen Leistungsqualität (Performance) bereit. Anders ausgedrückt wird ein System nach den folgenden Gesichtspunkten unterteilt: WOMIT (Ressource), WAS (Function), WIE (Design), WIE GUT (Performance), und WAS IST ZU BEACHTEN (Condition). Gilb teilt sämtliche Systemattribute in diese fünf Gruppen und gibt für jede einen Prozess und ein Vorgehen zu ihrer Spezifikation an.

Die Grundidee: Nichtfunktionale Anforderungen sind Performance Requirements

Gilb selbst benutzt den Begriff „nichtfunktionale Anforderung“ nicht, er spricht von Performance. Diese steht bei Planguage für Systemleistungen, die definieren, wie gut ein System ist, wie es die Außenwelt beeinflusst und wie es von den Stakeholdern wahrgenommen wird. Performance ist der Output im Sinne von Leistung, im Gegensatz zu Ressourcen, die den Input darstellen. Performance ist die Art und Weise, WIE GUT das System Funktionen ausführt, diese dem Nutzer zur Verfügung stellt und WIE GUT das System vom Nutzer unmittelbar wahrgenommen wird.

Es werden drei Arten von Performance unterschieden:

1. *Quality:* Definiert, WIE GUT das System arbeiten soll. Der Begriff ist hier im weitesten Sinne zu verstehen. Er beinhaltet alles, was nicht zu den anderen Arten gehört.
2. *Resource Savings:* Definiert, WIE VIEL im Vergleich zu einem anderen System oder einem anderen Vergleichswert eingespart werden soll. Dies kann zum Beispiel die Reduzierung der Kosten für bestimmte Transaktionen oder die Verkürzung der Durchlaufzeit einer bestimmten Aufgabe sein.
3. *Workload Capacity:* Definiert, WIE VIEL Leistung das System erbringt. Mit diesen Anforderungen sind unter anderem Kapazität und Durchsatz gemeint. Dazu zählt zum Beispiel die Durchschnittsgeschwindigkeit für das Ausführen einer bestimmten Funktion, die Speicherkapazität oder die vom System unterstützte Höchstzahl parallel arbeitender Anwender.

Performanceattribute zählen zu den skalaren Attributen, ihre konkrete Ausprägung kann einen von mehreren möglichen Werten annehmen und wird durch Messung bestimmt. Im Gegensatz dazu stehen die binären Attribute, die nur einen von zwei Werten annehmen können, nämlich in Bezug auf die Anforderungen „erfüllt“ oder „nichterfüllt“. Funktionale Anforderungen sind binär.

Spezifikation von skalaren Attributen

Bei der Spezifikation von skalaren Attributen sind die folgenden beiden Aspekte am wichtigsten:

1. die Sicherstellung der Mess- und Testbarkeit und
2. die Festlegung von Erfolg und Misserfolg.

Die Mess- und Testbarkeit wird dadurch sichergestellt, dass die Messmethode und der Maßstab zum Spezifikationszeitpunkt bereits festgelegt werden. Es werden Zielwerte und mögliche Einschränkungen definiert, anhand derer Erreichung Erfolg und Misserfolg der Umsetzung gemessen werden können. Dabei ist es zulässig, nicht genau einen Wert festzulegen, sondern im Sinne der Skalarität einen zulässigen Wertebereich.

Damit klar ist, wie skalare Attribute spezifiziert werden, liefert Planguage ein Regelwerk (Rules.SR)

und ein dazugehöriges Spezifikationstemplate, das im Kern die Spezifikation folgender Elemente vorsieht:

- *Tag:* Eindeutiger Name der Anforderung,
- *Definition:* Genaue Beschreibung der Anforderung,
- *Scale:* Maßeinheit, in der die Anforderung gemessen wird (bei Antwortzeiten zum Beispiel Sekunden),
- *Meter:* Messmethode (zum Beispiel Lasttest oder Nutzerbefragung),
- *Benchmark:* Vergleichswerte zur Orientierung (zum Beispiel aus Vorgänger- oder Konkurrenzsystemen),
- *Target:* Vorgabe der Anforderung, das angestrebte Ziel,
- *Constraint:* Zulässige Grenze der Anforderung nach unten oder oben.

Eine beliebte nichtfunktionale Anforderung ist die der Wartbarkeit. Hier ein vereinfachtes Beispiel, wie mit Planguage die Spezifikation dafür aussehen könnte:

- *Tag:* Wartbarkeit,
- *Definition:* Die Leichtigkeit, mit der ein System verändert oder erweitert werden kann,
- *Scale:* Stunden,
- *Meter:* Durchschnittliche Dauer für die Behebung eines Fehlers. Gemessen wird vom Beginn der Behebung bis zu dessen Lösung,
- *Target:* 2 h,
- *Constraint:* 4 h.

Das obige Beispiel lässt sich sicherlich noch weiter ausbauen und dadurch verbessern. Es soll jedoch vereinfacht zeigen, dass bei einer nichtfunktionalen Anforderung erstens klar sein muss, was alle Beteiligten darunter verstehen und zweitens wie sie operationalisiert wird.

Der Nutzen für die Praxis

Alles in allem ist Planguage ein höchst formalisiertes Vorgehensmodell. Das Glossar in der Fassung von [5] ist sehr umfangreich (ca. 115 Seiten und weit über 180 Konzepte). Der Einarbeitungsaufwand ist damit eher hoch und setzt gute analytische Fähigkeiten voraus. Der Aufwand dürfte für einige Leser zu hoch sein [2]. Die gesamte Spezifikation nach Planguage zu notieren, wird zudem ohne entsprechende Tools schnell unübersichtlich.

Allerdings kann für die Praxis aus Planguage der Nutzen gezogen werden, etwas über Performance und skalare Attribute zu lernen. Eine genau festgelegte Definition und Kategorisierung nichtfunktionaler Anforderungen steht nicht im Vordergrund. Wichtiger und damit entscheidend für den Erfolg ist die quantitative Spezifizierung.

In unserem Haus, einem der größeren IT-Beratungs- und Systemintegrationsunternehmen in Deutschland, sind wir pragmatisch vorgegangen. Unser Vorgehensmodell, das laufend überarbeitet wird, um den Softwareentwicklungsprozess gezielt zu verbessern, basiert auf dem Rational Unified Process [13, S. 27] und [9]. Der Bereich der nichtfunktionalen Anforderungen wurde mithilfe von Planguage komplett überarbeitet. Die Definition der Performance Requirements und die Kategorien Quality, Resource Savings und Workload Capacity wurden übernommen und das Template zur Spezifikation nichtfunktionaler Anforderungen entsprechend aktualisiert. Weiterhin ist die Sicherstellung der Test- und Messbarkeit und die Festlegung von Erfolg- und Misserfolg nun fester Bestandteil der Spezifikation und der Kontrolle nichtfunktionaler Anforderungen.

Literatur

1. Alexander IF, Stevens R (2002) Writing Better Requirements, 1. Aufl. Addison-Wesley, Boston
2. Alexander I (2005) Book Review: Competitive Engineering. http://i.f.alexander.users.btopenworld.com/reviews/gilb.htm, letzter Zugriff 5.11.2008
3. Antón A (1997) Goal Identification and Refinement in the Specification of Information Systems. PhD Thesis, Georgia Institute of Technology
4. Ebert C (2005) Systematisches Requirements Management, 1. Aufl. dpunkt, Heidelberg
5. Gilb T (2005) Competitive Engineering, 1. Aufl. Elsevier Butterworth Heinemann, Amsterdam
6. Glinz M (2008) On Non-Functional Requirements. 15th IEEE International Requirements Engineering Conference (RE'07), 15.–19.12.2007, Delhi, Indien. http://www.ifi.uzh.ch/rerg/fileadmin/downloads/publications/papers/RE07_Glinz.pdf, letzter Zugriff 16.10.2008
7. Jacobson I, Booch G, Rumbaugh J (1999) The Unified Software Development Process. Addison Wesley, Reading
8. Kotonya G, Sommerville I (1998) Requirements Engineering: Processes and Techniques. Wiley, Chichester
9. Kruchten P (2001) The rational unified process, 2. Aufl. Addison-Wesley, Reading
10. Ncube C (2000) A Requirements Engineering Method for COTS-Based Systems Development. PhD Thesis, City University London
11. Robertson S, Robertson J (2006) Mastering the Requirements Process. ACM Press books, 2. Aufl. Addison-Wesley, Boston
12. Rupp C & die SOPHISTen (2007) Requirements-Engineering und Management, 4. aktualisierte und erw. Aufl. Hanser, München
13. Singvogel R (2009) Wirtschaftlich und wirksam: Entwicklungsprozesse auf Basis des Eclipse Process Frameworks. In: Liggesmeyer P, Engels G, Münch J, Dörr J, Riegel N (Hrsg.) Proceedings of Software Engineering 2009 (SE-09), 2.–6.3.2009, Kaiserslautern, Germany, GI