Lernen aus dokumentierten Architektur-Entscheidungen

Andrea Herrmann, Barbara Paech
Universität Heidelberg, Fakultät für Mathematik und Informatik, Lehrstuhl Software Engineering, 69120 Heidelberg; {herrmann;paech}@informatik.uni-heidelberg.de

Indem man Architekturentscheidungen trifft, entscheidet man über die zukünftigen Eigenschaften des Systems. Bewertet man die Folgen einer Entscheidung, gewinnt man hieraus wertvolles Erfahrungswissen, das bei späteren Entscheidungen wieder verwendet werden kann. Dies wird unterstützt, indem man Architekturentscheidungen systematisch durchführt und nachvollziehbar dokumentiert. Hierzu gehören die klare Definition von Bewertungs- und Entscheidungskriterien sowie die quantitative Bewertung von Alternativen. Dieser Beitrag diskutiert, welche Formen von Erfahrungen innerhalb des Kreislaufs von Entscheidungen und Lernen beim Architekturdesign entstehen und wie sie wieder verwendet werden können. Dieses Lernen führt zu einer evolutionären Verbesserung des Entscheidungsprozesses und damit auch seiner Ergebnisse.

Schlüsselworte: Architektur, Design, Entscheidungen, Lernen

Motivation und Einleitung

Die kritischen Punkte im Softwareentwicklungsprozess sind diejenigen, an denen Entscheidungen getroffen werden. Denn hier wählt man zwischen mehreren möglichen Alternativen (das heißt Lösungen für die Gesamtarchitektur oder eine Komponente) und bestimmt damit die zukünftigen Eigenschaften des Systems und v.a. dessen Ausbaufähigkeit. Viele Entscheidungen lassen sich nach der Umsetzung der gewählten Alternative nicht oder schwer rückgängig machen. Tyree und Akerman [1] identifizieren wichtige Architekturentscheidungen, die besonders sorgfältig behandelt werden müssen, so: „To test a decision´s architectural significance, an architect should ask the following question: does this decision affect one or more system qualities (performance, availability, modifiability, security, and so on)? If so, an architect should make this decision and document it completely."

Wichtig ist es uns, zwischen dem Anforderungsraum und dem Lösungsraum zu unterscheiden. Anforderungen sind Wünsche. Sie stammen eventuell aus verschiedenen Quellen und werden oft unabhängig voneinander spezifiziert. Sie können einander widersprechen oder gar nicht erfüllbar sein. Im Lösungsraum jedoch wird eine realisierbare Architektur beschrieben. Diese Architektur erfüllt einige Anforderungen, andere dagegen vielleicht nicht. Zu den Anforderungen zählen auch Architekturanforderungen, d.h. Wünsche an die Architektur, an Komponenten oder deren Eigenschaften. Diese klingen oft wie Lösungsbeschreibungen, dürfen aber im Sinne eines systematischen Architekturdesigns auf keinen Fall mit Lösungen zu verwechselt werden. Auch wenn eine der Anforderungen verlangt, man solle eine bestimmte Standardsoftware verwenden, können die übrigen Anforderungen nach Prüfung von Alternativen nahe legen, dies letztlich nicht zu tun. Eine Architektur entsteht also nicht aus der Summe aller Architekturanforderungen, sondern nur als Folge von bewussten Entscheidungen (siehe auch [2]).

Beim Entscheiden werden Erfahrungen in vielen Formen verwendet und erzeugt. Erfahrung entsteht aus Schlussfolgerungen, die man aus früheren ähnlichen Entscheidungen zieht, beispielsweise durch nachträgliche Bewertung deren Folgen. Lernen bedeutet, Erfahrungen wieder zu verwenden. Im Bereich des Lernens geht man üblicherweise von Feedback-Schleifen aus [3-5]. In Abbildung 1 passen wir dieses Prinzip auf das Lernen aus Entscheidungen an. Ohne Lernen bestünde der Entscheidungsprozess (auch der Prozess der Architekturentscheidung) aus einer Folge von Aktivitäten (helle Ovale) ohne Rückmeldung, das heißt: (1) Frage, (2) Alternativenfindung, (3) Bewertung der Alternativen, (4) Entscheidung, d.h. der Auswahl der umzusetzenden Alternative auf Basis der Bewertungen, und (5) Umsetzung der gewählten Alternative.

Um innerhalb des Softwareentwicklungsprozesses Erfahrungen zu klassifizieren und dadurch wieder verwendbar zu machen, müssen Softwareentwicklungsmethoden eingesetzt werden, die systematische Entscheidungen und das Lernen aus Erfahrungen unterstützen. Hierzu muss eine solche Methode klar die Elemente einer Entscheidung definieren, dokumentieren und später auswerten. Außerdem muss sie projektübergreifend Erfahrungen in dieser einheitlichen Form zur Wiederverwendung zur

Verfügung stellen. Stellvertretend für solche Methoden diskutieren wir hier konkret anhand von MOQARE (=Misuse-Oriented Requirements Engineering) [6-8] für die Beschreibung von nichtfunktionalen Anforderungen und ICRAD (=Integrated Conflict Resolution and Architectural Design) [9] für den Entwurfprozess. Beide Methoden wurden an der Universität Heidelberg entwickelt und in Fallstudien eingesetzt. Erste Erfahrungen in Bezug auf Wiederverwendung und Lernen wurden hierbei gemacht. In diesem Beitrag stellen wir beide Methoden nicht vollständig dar, sondern beleuchten, wie sie Entscheidungen unterstützen und dokumentieren.

Der folgende Abschnitt beschreibt zunächst allgemein, welche wieder verwendbaren Erfahrungen bei Architekturentscheidungen verwendet werden und dabei entstehen. Anschließend diskutieren wir anhand eines Beispiels den Unterschied zwischen der intuitiven und der systematischen Erfahrungsnutzung.

Erfahrungen, die in Architekturentscheidungen verwendet werden und daraus entstehen

Das Lernen aus Erfahrung (dunkle Ovale in Abbildung 1) unterstützt und verbessert den Entscheidungsprozess an mehreren Stellen. Dies beginnt beim Schöpfen offener Fragen (1) aus einem **Fragenpool** und möglicher Alternativen (2) aus einem **Lösungspool**. Wir können davon ausgehen, dass erfahrene Mitarbeiter oder solche, die Zugriff auf die Erfahrungen anderer haben, mehr Alternativen entdecken. Die eigene Kenntnis von Lösungen wird erweitert durch Recherchen in der Fachliteratur, Diskussion mit Kollegen oder kreative Problemlösung während eines Projektes.

Erfahrungen fließen sowohl beim Bewerten der Alternativen (3) als auch beim Entscheiden (4) ein, und zwar in Form von Kriterien, Regeln und Richtwerten (z.B. Schwellenwerte für die Akzeptanz, Prognosewerte). (3) **Bewertungskriterien** für Alternativen sind in der Literatur nicht schwer zu finden. Es bieten sich je nach Perspektive Softwaremetriken und betriebs-wirtschaftliche Größen an. Die Kunst besteht hier vor allem darin, aus der Vielfalt die relevantesten auszuwählen und miteinander vergleichbar zu machen, z.B. durch Umrechnen in dieselbe Maßeinheit.

Regeln und Methoden, um Kosten und die Werte anderer Bewertungskriterien vorherzusagen, gibt es in der Fachliteratur. Hinzu kommen firmeninterne Erfahrungswerte.

(4) Die **Kriterien für die Entscheidung** sind in ICRAD der Nettowert und das Nutzen-Kosten-Verhältnis, das heißt Größen, die mehrere Bewertungskriterien berücksichtigen. In einfachen Fällen dienen die Bewertungskriterien unmittelbar als Entscheidungskriterien, beispielsweise wenn nur die Stakeholderpräferenzen eine Rolle spielen wird die von den Stakeholdern bevorzugte Alternative gewählt. Ansonsten sind kombinierte Größen üblich, sowohl in der Betriebswirtschaft als auch der Fachliteratur zur Anforderungspriorisierung [10,11].

Fragen- und Lösungspool, Kriterien, Regeln und Prognosewerte entstehen aus Erfahrungen und werden durch die Bewertung weiterer Entscheidungen und deren Folgen verbessert, z.B. durch den Vergleich der bei oder nach der Umsetzung tatsächlich realisierten mit den erwarteten Werten oder Ergebnissen.

Das Lernen beginnt jedoch bereits während des Entscheidungsprozesses. Immer wieder überprüft man Prognosewerte und Zwischensummen der Bewertungs- und Entscheidungskriterien auf ihre Plausibilität. Zweifel an den Zwischenergebnissen führen dazu, dass Abschätzungen und deren Annahmen überprüft werden und noch offene Fragen entdeckt. Auch wenn Qualitätsziele von Beteiligten unausgesprochen als selbstverständlich angenommen werden, kann dies bei der Nutzenanalyse entdeckt werden.

Intuitive Erfahrungsnutzung bei der Architekturentscheidung

Die intuitive Erfahrungsnutzung funktioniert im Alltag normalerweise gut. Allerdings erschwert sie eine für andere nachvollziehbare Begründung einer Entscheidung sowie die unvoreingenommene Wiederverwendung eigener und fremder Erfahrungen.

(1) Als Beispiel wählen wir eine ganz grundlegende Architekturentscheidung, nämlich die bezüglich der Zerlegung des Systems in Subsysteme.

(2) Dem Softwarearchitekten fallen hierzu als Alternativen vielleicht eine Client-Server-Architektur und ein Peer-to-peer-Netzwerk ein.

(3) Er sagt sich, dass er mit Client-Server am meisten Erfahrung hat, und schließlich wird er derjenige sein, der System später warten muss. Dies ist bei einer unbekannten Technologie aufwändiger. Außerdem fällt ihm ein, dass ein Kollege ihm einst von einem Projekt erzählte, in dem sie mit dem Peer-to-peer Ansatz Probleme wegen des komplexen Kontrollflusses hatten.

(4) Damit fällt seine Entscheidung auf die Client-Server-Alternative.

Solche Überlegungen setzen implizit Erfahrung voraus in Form von **Fragenpool** und **Lösungspool**. Der Designer könnte ja auch ohne Entscheidung immer nur Client-Server-Systeme entwickeln. Wir werden sehen, dass dem Softwarearchitekten durchaus mehr als zwei Alternativen zur Verfügung gestanden hätten. Nehmen wir an, er habe auch von anderen Lösungen gehört, jedoch nichts Gutes. Da sich schlechte Erfahrungen effektiver verbreiten als gute, kann dieser Eindruck verzerrt sein. Hinter der Beschränkung auf Bekanntes oder besonders sinnvoll Erscheinendes steckt bereits Erfahrung in Form von **Bewertungs- und Entscheidungskriterien**, hier implizit die eigene oder fremde gute Erfahrung mit einer Alternative ist. Worin diese gute Erfahrung besteht, ist hier allerdings nur bruchstückhaft und subjektiv definiert. Manche

Softwarearchitekten finden besonders die Performance wichtig, andere die Wartbarkeit, und der Projektleiter legt mehr Wert auf die Entwicklungskosten.

Die Schritte und Elemente, die laut Abbildung 1 zu einer Entscheidung und dem Lernen aus Entscheidungen gehören, finden wir also beim intuitiven Entscheiden durchaus wieder. Allerdings mangelt es durchgängig an Vollständigkeit und Nachvollziehbarkeit.

Systematische Erfahrungsnutzung bei der Architekturentscheidung mit MOQARE und ICRAD

Obwohl bei der intuitiven Erfahrungsnutzung im Architekturdesign die Elemente der Feedback-Schleife aus Abbildung 1 auftreten, lässt sich der Prozess durch systematische Softwareentwicklungsmethoden und Erfahrungssammlung verbessern. MOQARE und ICRAD dienen hier dazu, konkret aufzuzeigen, wie solche Methoden dies unterstützen können.

(1) **Fragenpools** für das Architekturdesign findet man in der Fachliteratur in der Form von Architekturmodellen oder Designprozessen. Meist geht man top-down vor, vom Groben zum Feinen, da die grundlegenden Entscheidungen die Alternativen auf niedrigeren Ebene und deren Bewertung beeinflussen. Nach Bruegge und Dutoit [12] sind beim Architekturdesign von oben nach unten folgende Arten von Entscheidungen zu treffen: Zerlegung des Systems in Subsysteme, Mapping der Subsysteme auf Hardware- und Software-Ressourcen, Art der Speicherung persistenter Daten, globaler Daten- und Kontrollfluss. Teil von ICRAD ist auch ein Verfahren zur Bewertung der Auswirkung von Entscheidungen derselben Ebene, so dass diese in einer sinnvollen Reihenfolge durchgeführt werden können, beginnend mit denjenigen mit der weitest gehenden Auswirkung auf die Architektur und die anderen Entscheidungen.

(2) **Alternativen** für jede Art von Entscheidungen sowie deren Vor- und Nachteile werden ebenfalls durch die Fachliteratur diskutiert, und somit durch sie Erfahrungen zur Verfügung gestellt.

Bei unserem Beispiel der Entscheidung zwischen Client-Server-Architektur und Peer-to-peer-Netzwerk wird bezüglich der Zerlegung des Systems in Subsysteme entschieden. Für diese Frage geben Bruegge und Dutoit [12] die folgenden alternativen Lösungen an: repository style, client-server style, peer-to-peer, model/view/controller style, three tier, four tier, und pipe and filter. Diese Literaturquelle dient uns also sowohl als Fragenpool als auch als **Lösungspool** (und liefert auch Bewertungen der Alternativen).

(3) ICRAD verwendet vier **Bewertungskriterien** für Alternativen: Nutzen, Risiko, Kosten und Wartungskosten. Nach unseren Erfahrungen können diese vier viele andere Bewertungskriterien wie Dringlichkeit, für die Realisierung nötige Kalenderzeit, Komplexität, und so weiter [13] ebenfalls berücksichtigen, wenn die nichtfunktionalen Anforderungen mit MOQARE [6-8] beschrieben werden. Sowohl Nutzen als auch Risiko von Alternativen schätzen wir auf der Basis der Anforderungen, die durch eine Entscheidung betroffen sind, d.h. entweder durch die Entscheidung erfüllt oder nicht erfüllt werden können. Der Nutzen setzt sich aus zwei Anteilen zusammen: erstens aus dem der funktionalen und nichtfunktionalen Anforderungen, die die Stakeholder definiert haben und deren Nutzen sie am besten selbst vorhersagen; zweitens dem Nutzen und dem Grad der Erfüllung der definierten Qualitätsziele bzw. der sie unterstützenden Anforderungen. Die Gewichtungen dieser Ziele und wie gut sie (voraussichtlich) erfüllt werden, hängt vom Kontext und den Stakeholdern ab. Die Vorhersage von Nutzen und Risiken gehört zu den schwierigsten Aufgaben des gesamten Prozesses. Hierbei müssen Abhängigkeiten zwischen Anforderungen berücksichtigt werden und Abschätzungen dürfen nur in Bezug auf ein Referenzsystem erfolgen [13]. Beim Abschätzen von Wahrscheinlichkeiten, Nutzen und Schäden sind Erfahrungswerte sehr wichtig.

Als Ausgangspunkt für die Bewertung des Nutzens nichtfunktionaler Anforderungen einer Software kommt eine Vielzahl von Qualitätsattributen in Frage, wie diejenigen der Norm ISO 9126 [14]. Auf der Basis dieser Qualitätsattribute lassen sich Qualitätsziele definieren im Stil von: „Das System soll einfach wartbar sein.“ Oder: „Der Datenfluss soll zuverlässig sein.“ Diese verschiedenen Qualitätskriterien lassen sich nur miteinander vergleichen und gegeneinander abwägen, wenn sie in derselben Maßzahl quantifiziert sind. Darum weisen wir (vereinfacht dargestellt) jedem Qualitätsziel einen Nutzen-Wert zu, der misst, wie gut es die Ziele des Gesamtsystems unterstützt.

MOQARE hilft dann zu konkretisieren, welche Faktoren das Erreichen von Qualitätszielen unterstützen oder schädigen. (Hier werden von MOQARE nur die für die Bewertung von Architekturalternativen wichtigen Elemente beschrieben.) Die schädlichen Faktoren werden durch Misuse Cases (Fehlnutzungsszenarien) beschrieben, die möglicherweise die Erreichung eines Qualitätsziels bedrohen. Es werden Gegenmaßnahmen definiert, die diese Misuse Cases verhindern, abschwächen oder zumindest entdecken sollen und somit die Qualitätsziele unterstützen. Die Zusammenhänge werden in MOQARE grafisch übersichtlich daargestellt. MOQARE stellt Erfahrungen anderer in einheitlicher Form zur Verfügung, u.a. als Checklisten für die Ermittlung von Misuse Cases und Gegenmaßnahmen [8].

Die Misuse Cases und Gegenmaßnahmen erlauben eine detailliertere Bewertung der Alternativen in Bezug auf deren Erfüllung der Qualitätsziele: Wir berechnen den Nutzen (benefit) einer Alternative in Bezug auf ein

Qualitätsziel aus dem Produkt des Erfüllungsgrad dieses Qualitätsziels und dessen Nutzen.

Im vorigen Kapitel waren unserem Architekturdesigner spontan zwei Qualitätsziele eingefallen: die Zuverlässigkeit des Datenflusses und die einfache Wartbarkeit. Ein Blick in die ISO 9126 könnte ihn darauf bringen, dass die Performance ebenfalls ein Qualitätsziel ist, das sowohl wichtig ist als auch geeignet, die Alternativen zu unterscheiden. Wenn wir annehmen, dass in einem konkreten Kontext (z.B. bestimmte Firma, Projekt) die Zuverlässigkeit des Datenflusses wichtiger ist als die die beiden anderen Qualitätsziele, gewichten wir z.B. den Nutzen der Zuverlässigkeit mit 3 Punkten und den der Wartbarkeit sowie der Performance mit jeweils 2 Punkten. Sind also alle drei Qualitätsziele zu 100% erfüllt, erreicht das System 7 Punkte. Die Funktionalität sei weitere 7 Punkte wert (in beiden Alternativen jeweils vorhanden). Es kann jede beliebige Skala verwendet werden, die den Beteiligten sinnvoll erscheint und während des Entscheidungs- und Lernprozesses durchgehalten werden kann.

Ein Misuse Case wird ähnlich wie ein Use Case beschrieben. Der Kürze der Darstellung halber betrachten wir hier nur sechs stichwortartige Misuse Cases, jeweils zwei pro Qualitätsziel:

- Zuverlässigkeit des Datenflusses
  - Die Komplexität des Kontrollflusses macht die fehlerfreie Entwicklung schwierig.
  - Mangelhafte Interoperabilität der Subsysteme (z.B. durch Programmierfehler) oder Deadlock führt zu Datenverlust oder Datenverfälschung.
- einfache Wartbarkeit
  - Die Änderung eines Subsystems führt zu bedeutenden oder mehrfachen Änderungen an anderen Subsystemen.
  - Innerhalb der Firma ist wenig Erfahrung mit dieser Technologie vorhanden.
- Performance
  - Ein Flaschenhals im Architekturstil führt im Normalbetrieb zu Performanceverlusten.
  - Hohe Datenmengen bereiten zusätzliche Performanceprobleme.

Hierbei wurden durchaus auch die intuitiven Entscheidungskriterien aus dem vorigen Abschnitt verwendet, einschließlich der Überlegung, dass die Wartung problematisch ist, wenn in der Firma wenig Erfahrung mit dieser Technologie vorhanden ist.

Der Nutzen der Qualitätsziele wird durch die Misuse Cases gemindert, falls sie eintreten. Bewertet wird ein Misuse Case anhand seines Risikos (vgl. [15]) in Bezug auf ein Qualitätsziel, d.h. dem Produkt der Wahrscheinlichkeit seines Eintretens und dem (am Qualitätsziel) angerichteten Schaden. Der effektive Nutzen (effective benefit) einer Alternative berechnet sich dann aus *Nutzen – Risiko*, wobei sich das Risiko (risk). Während der Nutzen von der Architektur im Beispiel nicht abhing, gilt dies für die Risiken sehr wohl (siehe Tabelle 1). Um eine vorschnelle unfundierte Entscheidung zu vermeiden, sollten eigentlich alle sieben genannten Alternativen berücksichtigt werden. Allerdings soll die Darstellung des Beispiels in dieser Veröffentlichung nicht zu umfangreich werden und wir beschränken uns auf die client-server (I) und die peer-to-peer (II) Alternativen.

Tabelle 1: Abschätzung der Risiken der Misuse Cases und des effektiven Nutzens pro Qualitätsziel

Qualitätsziel (QZ) oder Misuse Case (MUC)	**Alternative (I)**	**Alternative (II)**
Komplexität des Kontrollflusses (MUC)	0 · 3 = 0	1% · 3 = 0,03
Mangelhafte Interoperabilität (MUC)	0,01% · 3 = 0,0003	0,01% · 3 = 0,0003
Zuverlässigkeit des Datenflusses (QZ)	3 – 0 – 0,0003 = 2,9997	3 – 0,03 – 0,0003 = 2,9697
Änderung eines Subsystems führt zu[…]Änderungen an anderen (MUC)	10% · 1 = 0,1	10% · 1,5 = 0,15
wenig Erfahrung mit Technologie (MUC)	0% · 1 = 0	100% · 1 = 1
einfache Wartbarkeit (QZ)	2 – 0,1 – 0 = 1,9	2 – 0,15 – 1 = 0,85
Flaschenhals (MUC)	1% · 1 = 0,01	0,01% · 1 = 0,0001
Hohe Datenmengen (MUC)	20% · 1 = 0,2	20% · 1,5 = 0,3
Performance (QZ)	2 – 0,01 – 0,2 = 1,79	2 – 0,0001 – 0,3 = 1,6999
Risiko der Alternative = Summe der Risiken der MUC	0 + 0,0003 + 0,1 + 0 + 0,01 + 0,2 = 0,3103	0,03 + 0,0003 + 0,15 + 1 + 0,0001 + 0,3 = 1,4804
Summe der effektiven Nutzen der QZ	2,9997 + 1,9 + 1,79 = 6,6897	2,9697 + 0,85 + 1,6999 = 5,5196

Wir verzichten hier der Übersichtlichkeit halber auf die Diskussion von Gegenmaßnahmen gegen die Misuse Cases. Diese reduzieren das Risiko jeweils und verbessern die Erreichung der Qualitätsziele. Wegen Abhängigkeiten zwischen Risiken und Nutzen, Anforderungen und Misuse Cases, ist es streng genommen nicht erlaubt, Risiken oder Nutzen jeweils zu summieren [13]. Dies ist näherungsweise legitim, je geringer diese Abhängigkeiten. Misuse Cases, die stark voneinander abhängen (beispielsweise dieselbe Ursache haben) können zusammengefasst und gemeinsam behandelt werden.

In die Schätzungen gehen nach unserer Erfahrung detaillierte Annahmen über das Systemumfeld, die Benutzung und auch die Architektur ein. Außerdem muss man eine Zeitdauer festlegen, in Bezug auf die alle zeitabhängigen Größen geschätzt werden. Diese Annahmen müssen ebenfalls dokumentiert werden. Da wir kein konkretes System im Sinn haben, sondern nur das Prinzip der Methode beschreiben wollen, sind die Abschätzungen in den Tabellen 1 und 3 als rein hypothetisch zu verstehen.

(4) Die Qualitätsziele beziehungsweise der hieraus abgeleitete effektive Nutzen der Alternativen sind nicht die endgültigen Entscheidungskriterien. Sonst würden wir nun Alternative (I) wählen. Es kann jedoch Argumente gegen diese Lösung geben, beispielsweise die Server-Kosten. Ein Server verursacht höhere Anschaffungs- und Installationskosten (cost), aber auch eventuell geringere Wartungskosten (complexity cost), als ein Rechner im peer-to-peer-Netzwerk. Die Kosten hängen natürlich von der konkreten technischen Realisierung ab, und die Wartungskosten zusätzlich von der erwarteten Lebensdauer der Software. Die Ergebnisse könnten ausfallen wie in Tabelle 3.

Um die vier Bewertungskriterien Nutzen (benefit), Risiko (risk), Realisierungskosten (cost) und Wartungskosten (complexity cost) gegeneinander abzuwägen, verwendet ICRAD die **Entscheidungskriterien** „Nettowert“ (net value = effective benefit – total cost) und „Nutzen-Kosten-Verhältnis“ (effective benefit / total cost) der Alternativen. Diese sollen beide möglichst große Werte haben. ICRAD stellt sowohl die Bewertungskriterien der Alternativen als auch die zwei Entscheidungskriterien tabellarisch dar. Tabelle 2 zeigt die Vorlage und Tabelle 3 die Schätzwerte aus dem Beispiel. Die Spalte „Difference“, in der jeweils der Unterschied zwischen den Werten der beiden Alternativen berechnet wird, unterstützt den Vergleich.

Wenn die beiden Kriterien Nettowert und Nutzen-Kosten-Verhältnis nicht dieselbe Alternative unterstützen, stellt der in der Vorlage (Tabelle 2) rechts unten stehende Wert ein weiteres sehr gutes Entscheidungskriterium dar [9]. Wir schreiben ihn $\Delta TB/\Delta TC$, und er misst das Verhältnis zwischen dem Unterschied des effektiven Nutzens der beiden Alternativen und dem Unterschied deren Gesamtkosten.

Diese Vorlage unterstützt die Entscheidungsfindung und dokumentiert sie. Die tabellarische Vorlage dient hierbei nur als Zusammenfassung der Ergebnisse des Entscheidungsprozesses und muss durch eine ausführliche Dokumentation der einzelnen Faktoren (z.B. Risiken) und der Annahmen ergänzt werden, die den Schätzungen zugrunde liegen.

Beim Vergleich der beiden Alternativen in Tabelle 3 sieht man, dass (II) geringeren Nutzen bringt bei höheren Kosten. Sowohl der Nettowert als auch das Nutzen-Kosten-Verhältnis sind bei Alternative (I) trotz der höheren Anschaffungs- und Installationskosten höher und daher besser. Daher wird (I) gewählt.

Tabelle 2: Tabellenvorlage zum Vergleich von Alternativen

	Alt. (I)	**Alt. (II)**	**Difference (II)-(I)**
Cost	C1	C2	C2-C1
Complexity Cost	CC1	CC2	CC2-CC1
Risk	R1	R2	R2-R1
Benefit	B1	B2	B2-B1
Effective benefit	B1-R1	B2-R2	(B2-R2)-(B1-R1) $= \Delta TB$
Total cost	C1+CC1	C2+CC2	(CC2- CC1) +(C2-C1) $= \Delta TC$
Net value	(B1-R1)-(C1+CC1)	(B2-R2)-(C2+CC2)	(B2-R2)-(C2+CC2) -(B1-R1) +(C1+CC1)
Effective Benefit/ total cost	(B1-R1) / (C1+CC1)	(B2-R2) / (C2+CC2)	$\Delta TB/\Delta TC$ = [(B2-R2) -(B1-R1)] / [(CC2-CC1) +(C2-C1)]

Tabelle 3: Beispielhafter Vergleich zweier Alternativen (Risikoschätzungen aus Tabelle 1, weitere Erklärungen siehe Text)

	Alt. (I)	**Alt. (II)**	**Difference (II)-(I)**
Cost	4,5	4	-0,5
Complexity Cost	3	4	1
Risk	0,3103	1,4804	1,1701
Benefit	14	14	0
Effective benefit	13,6897	12,5196	-1,1701
Total cost	7,5	8	0,5
Net value	6,1897	4,5196	-1,6701
Effective Benefit/ total cost	1,8253	1,5650	-2,3402

Zusammenfassung und Diskussion

Dieser Beitrag beschreibt, wie beim Architekturdesign das Treffen von Entscheidungen unterstützt und dokumentiert werden kann, und wie Erfahrungen dabei genutzt und daraus gewonnen werden.

Ein systematisches Vorgehen beim Entscheiden und dessen Dokumentation bietet ein Rahmenwerk, um Erfahrungen zu identifizieren, zu klassifizieren und

wieder zu verwenden. Konkret diskutiert wird anhand der Methoden MOQARE für die Beschreibung von nichtfunktionalen Anforderungen und ICRAD für die Unterstützung des Architekturdesigns. Diese können als stellvertretend gesehen werden für andere systematische Methoden. Als projekt- und technologieübergreifend wieder verwendbar haben sich Fragen und Lösungen erwiesen, so dass sie in Fachliteratur gesammelt und weitergegeben werden können (siehe [12] oder die Checklisten in [8]), sowie Bewertungs- und Entscheidungskriterien. Richtwerte und Regeln hängen stark vom Kontext ab, dass sie nur in ähnlichen Projekten wieder verwendet werden können. Daher ist es ratsam, dass jede Organisation diese Daten in wieder verwendbarer Form sammelt.

Da die quantitative Bewertung der Alternativen stark von Umfeld und Annahmen abhängt, könnte man an deren Sinn sowie der darauf basierenden Entscheidung zweifeln. Wir gehen davon aus, dass diese Abhängigkeit aus der Komplexität der Sache resultiert und nicht vermieden werden kann.

Trotzdem macht der Prozess der Abwägung Sinn, da hierbei implizite Ziele, Kriterien und Annahmen bewusst werden, diskutiert und dokumentiert werden, und die an der Softwareentwicklung Beteiligten ihre Entscheidungen sowie deren Folgen reflektieren. Dies führt bereits zu Erfahrungsnutzung und Lernen.

Bewährt hat sich auch die Dokumentation von Entscheidungsgründen, um die erwarteten und tatsächlichen Folgen einer Entscheidung vergleichen zu können. Den Aufwand für diese Dokumentation kann man durch die klare Definition des Vorgehens und der Entscheidungskriterien gering halten und auf die wichtigsten Entscheidungen [1] beschränken.

Referenzen

[1] Tyree, J., Akerman, A.: Architecture Decisions: Demystifying Architecture. IEEE Software, March/ April (2005) 19-27

[2] Bass, L., Clements, P., Nord, R.L., Stafford, J.: 12. Capturing and Using Rationale for a Software Architecture, in: Rationale Management in Software Engineering, Dutoit, A.H., McCall, R., Mistrík, I., Paech, B. (Eds.), Springer, Berlin (2006)

[3] Aamodt, A., Plaza, E.: Case-Based Reasoning: Foundational Issues. Methodological Variations and System Approaches AI Communications 17(1) (1994)

[4] Basili, V.R., Caldiera, G., Rombach, H.D.: The Experience Factory. Encyclopedia of Software Engineering - 2 Volume Set (1994) 469-476

[5] Louridas, P., Loucopoulos, P.: A Generic Model for Reflective Design. ACM Trans. Software Eng. and Methodology 9(2) (2000) 199-237

[6] Herrmann, A., Paech, B.: MOQARE = Misuse-oriented Quality Requirements Engineering - Über den Nutzen von Bedrohungsszenarien beim RE von Qualitätsanforderungen. Softwaretechnik-Trends 26(1) (2006) 13-14

[7] Herrmann, A., Paech, B.: Quality Misuse. REFSQ - Workshop on Requirements Engineering for Software Quality, Foundations of Software Quality, Essener Informatik Berichte (2005) 193-199

[8] Herrmann, A., Paech, B.: Software Quality by Misuse Analysis. Technical Report, http://www-swe.informatik.uni-heidelberg.de/research/publications/SIKOSA_WP2005-AH-1_V1.4withoutCaseStudy.pdf

[9] Herrmann, A., Paech, B., Plaza, D.: ICRAD: An Integrated Process for Requirements Conflict Solution and Architectural Design. IJSEKE 16(6) (2006)

[10] Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. IEEE Software 14(5) (1997) 67–74

[11] Srikanth, H., Williams, L.: On the Economics of Requirements-Based Test Case Prioritization. Proceedings of EDSER (2005)

[12] Bruegge, B., Dutoit, A.H.: Object-Oriented Software Engineering - Using UML, Patterns, and Java, Prentice Hall, NJ (2004)

[13] Herrmann, A., Paech, B.: Benefit Estimation of Requirements Based on a Utility Function. REFSQ - Workshop on Requirements Engineering for Software Quality, Foundations of Software Quality, Essener Informatik Berichte (2006)

[14] International Standard ISO/IEC 9126. Information technology -- Software product evaluation -- Quality characteristics and guidelines for their use

[15] International Standards Organization, ISO: Risk management – Vocabulary – Guidelines for use in standards, ISO Guide 73, Geneva (2002)

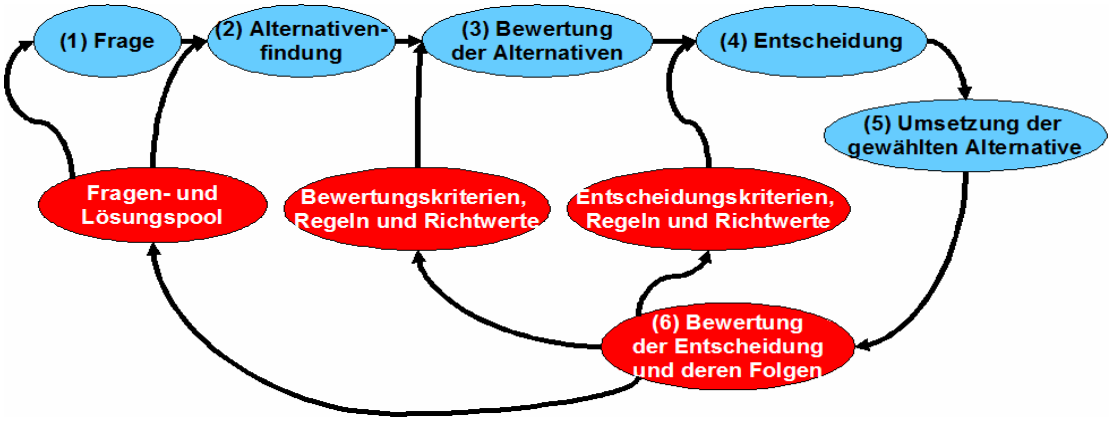

Abbildung 1: Feedback-Schleife:

Lernen aus Entscheidungen und für Entscheidungen