February 2007

Interaktive Entscheidungsunterstützung für die Auswahl von Software-Komponenten bei mehrfachen Zielsetzungen

Thomas Neubauer
Secure Business Austria - Security Research, neubauer@securityresearch.at

Christian Stummer
Universität Wien, christian.stummer@univie.ac.at

In: Oberweis, Andreas, u.a. (Hg.) 2007. *eOrganisation: Service-, Prozess-, Market-Engineering*; 8. Internationale Tagung Wirtschaftsinformatik 2007. Karlsruhe: Universitätsverlag Karlsruhe

ISBN: 978-3-86644-094-4 (Band 1)
ISBN: 978-3-86644-095-1 (Band 2)
ISBN: 978-3-86644-093-7 (set)

Interaktive Entscheidungsunterstützung für die Auswahl von Software-Komponenten bei mehrfachen Zielsetzungen

Thomas Neubauer

Secure Business Austria - Security Research
1040 Wien
neubauer@securityresearch.at

Christian Stummer

Institut für Betriebswirtschaftslehre
Universität Wien
1210 Wien
christian.stummer@univie.ac.at

Abstract

In der betrieblichen Praxis kommt der komponentenbasierten Software-Entwicklung hoher Stellenwert zu. Angesichts mehrfacher Zielsetzungen und vielfältiger Nebenbedingungen ist dabei insbesondere die Auswahl der „besten“ Kombination von Komponenten ein nicht-triviales Entscheidungsproblem. Bislang wurden hierfür vor allem die Nutzwertanalyse bzw. der Analytic Hierarchy Process zur Entscheidungsunterstützung vorgeschlagen, wobei aber beide eine Reihe von Unzulänglichkeiten aufweisen. Diese Arbeit will dazu nunmehr eine Alternative anbieten. Darin werden in einem ersten Schritt zunächst (zulässige) Pareto-effiziente Kombinationen von Software-Komponenten berechnet und die Entscheidungsträger dann im zweiten Schritt interaktiv bei der Suche nach jener Variante unterstützt, die einen Ziele-Mix in Aussicht stellt, der den jeweiligen individuellen Präferenzen am besten entspricht. Das neue Verfahren zeichnet sich im Vergleich zu herkömmlichen Ansätzen insbesondere durch den Verzicht auf umfangreiche a priori Präferenzinformationen (wie z.B. Zielgewichtungen) aus. Darüber hinaus kann es ohne großen Anpassungsaufwand in bestehende Vorgehensmodelle zur Auswahl von Software-Komponenten integriert werden.

1 Einleitung

Die komponentenbasierte Software-Entwicklung unterscheidet sich vom traditionellen Vorgehen insbesondere dadurch, dass (bestehende) Komponenten als Grundlage für die Entwicklung komplexer Softwarelösungen genutzt werden. Tatsächlich kommt ihr heutzutage hoher Stellenwert zu [Ruhe2002], da in immer kürzeren Abständen qualitativ hochwertige und zuverlässige Software auf den Markt gebracht werden muss. Zudem steigen die funktionalen Anforderungen an Software, so dass insbesondere kleinere Unternehmen bei der Erfüllung ihrer Aufträge zunehmend davon abhängig sind, auf vorhandene Komponenten zurückgreifen zu können und das Produkt nicht in allen Details von Grund auf neu entwickeln zu müssen [Alves2003]. Empirische Studien [Alves2003, Basili1996] zeigen ferner, dass durch den Rückgriff auf bewährte Komponenten Fehler im Gesamtsystem wesentlich reduziert werden. Und schließlich wird die komponentenbasierte Software-Entwicklung auch durch Technologien wie CORBA, JavaBeans/EJB, DCOM/ActiveX oder .Net sowie die Verfügbarkeit von verschiedenen Tools für die Konfiguration und den Einsatz solcher Lösungen vorangetrieben [Andrews2005].
Allerdings müssen Software-Produkte in der Regel an spezifische Anforderungen angepasst werden. Dementsprechend spielt bei der komponentenbasierten Software-Entwicklung die Auswahl der „richtigen“ Komponenten eine wesentliche Rolle mit Auswirkungen auf alle weiteren Phasen des Entwicklungszyklus. Ineffiziente Entscheidungen haben daher nicht nur Einfluss auf Korrektheit und Zuverlässigkeit der komponentenbasierten Anwendung, sondern können auch zu massiven Kostensteigerungen in der Entwicklung und/oder der nachfolgenden Wartung führen [Maiden1998,Ruhe2002,Ruhe2003].

In der Literatur finden sich zahlreiche Vorgehensmodelle für die Evaluierung und Auswahl von Software-Komponenten (z.B. OTSO von [Kontio1995]). Nahezu alle diese Ansätze berücksichtigen mehrfache Zielsetzungen (wie Kosten, Kompatibilität, Einfachheit der Installation, usw.), wobei die meisten Autoren entweder eine Nutzwertanalyse (Weighted Scoring Method; WSM) oder den Analytic Hierarchy Process (AHP) zur Entscheidungsfindung empfehlen (vgl. u.a. [Alves2003,Maiden1998,Navarrete2005,Ncube2002,Wanyama2005]). Tatsächlich weisen aber beide Verfahren wesentliche Schwachstellen auf, wie insbesondere den Bedarf an umfangreichen Informationen über die a priori Präferenzen der Entscheidungsträger bei der Nutzwertanalyse, die kombinatorische Explosion der paarweisen Vergleiche bei der Verwendung des

AHP oder problematische Annahmen über die Form der Nutzenfunktion (für die regelmäßig lineare Nutzenverläufe unterstellt werden). Des Weiteren bieten solche Ansätze dem Entscheidungsträger lediglich eine einzelne Lösung, während es ein interaktives Verfahren erlauben würde, unterschiedliche Szenarien zu erkunden und zu analysieren bzw. aktiv am Entscheidungsprozess teilzunehmen und ihn zu kontrollieren.

In dieser Arbeit stellen wir einen entsprechenden, zweiphasigen Ansatz vor. Die erste Phase widmet sich der Ermittlung von Lösungen (d.h. Kombinationen von Software-Komponenten), die einerseits gegebene Nebenbedingungen (wie Ressourcenbeschränkungen oder Abhängigkeiten zwischen zwei oder mehreren Komponenten) erfüllen und andererseits Pareto-effizient[1] hinsichtlich der gegebenen Zielsetzungen sind. In der zweiten Phase werden Entscheidungsträger bei der interaktiven Erkundung des solcherart bestimmten Lösungsraums unterstützt, bis sie die für sie individuell „beste“ Zusammenstellung von Komponenten mit dem für sie attraktivsten Mix an Zielwerten gefunden haben. Der Ansatz kann im Übrigen problemlos in bestehende Vorgehensmodelle für die komponentenbasierte Software-Entwicklung integriert werden. Diese Arbeit bietet nun im Anschluss zunächst einen Überblick zum Forschungsstand hinsichtlich der Auswahl von Software-Komponenten bei mehrfachen Zielsetzungen. Danach beschreiben wir schrittweise unseren Vorschlag zur interaktiven Entscheidungsunterstützung. Die Arbeit schließt letztlich mit einem Resümee und einem Ausblick auf in diesem Zusammenhang interessante weiterführende Forschungsfragen.

2 Die Auswahl von Software-Komponenten im Überblick

Im Folgenden liegt das Augenmerk auf Software-Komponenten, die im Rahmen einer Wiederverwendung Teil eines neuen Softwareprodukts werden können und dazu in der Regel schon von vornherein nicht spezifisch für ein bestimmtes Projekt erstellt worden sind (für eine Diskussion vgl. [Torchiano2004]). Die komponentenbasierte Software-Entwicklung zielt nun darauf ab, mit Hilfe solcher mehrfach nutzbarer Komponenten komplexe Software-Projekte in kürzerer Zeit bzw. mit geringerem Budget durchführen zu können. In der Praxis werden hierbei allerdings oftmals die Risiken unterschätzt, die mit der Evaluierung, Auswahl und Integration der Komponenten verbunden sind, so dass mitunter beträchtliche Verzögerungen bzw. Budget-

[1] Eine Lösungsalternative gilt als „Pareto-effizient“, wenn keine andere (zulässige) Lösung existiert, die in allen betrachteten Zielsetzungen zumindest gleich gut und in mindestens einer Zielsetzung besser abschneidet.

überschreibungen für die Entwicklung oder Wartung der Systeme zu beobachten sind [Tran1997].

In der Vergangenheit wurden mehrere Vorgehensmodelle zur Strukturierung der komponentenbasierten Software-Entwicklung vorgeschlagen. Obgleich sie sich über weite Strecken stark ähneln, gibt es bislang noch kein allgemein als Standard akzeptiertes Vorgehen [Ruhe2002]. Andrews unterscheidet dazu zwischen „architektur-“ und „anforderungsgetriebenen“ Vertretern, wobei die erste Gruppe etwa OTSO [Kontio1995], BAREMO [LozanoTello2002], oder STACE [Kunda2003] umfasst und SCARLET [Maiden2002] (als Nachfolger von PORE [Maiden1998]) sowie CRE [Alves2001] zur zweiten Gruppe zählen [Andrews2005].

Gemeinsam ist den Verfahren, dass an irgendeiner Stelle verschiedene Lösungsalternativen bewertet werden müssen. Eindimensionale Kennzahlen aus der Finanzwirtschaft wie etwa der Kapitalwert greifen aber für die komplexe Bewertung von IT-Investitionen regelmäßig zu kurz, so dass hierfür bislang die Nutzwertanalyse oder der AHP eingesetzt worden sind. Beide Verfahren erfordern jedoch die Bekanntgabe der Form der Nutzenfunktion (wobei meist der Einfachheit halber Linearität unterstellt wird) sowie von Gewichtungen für jedes zu berücksichtigende Zielkriterium. Die Nutzwertanalyse setzt dabei voraus, dass (i) sich die Entscheidungsträger von vornherein vollständig über alle Gewichtungen im Klaren sind, ohne jemals zuvor eine Lösungsalternative gesehen zu haben, und (ii) auch bereit sind, diese Präferenzen vollständig und vorbehaltlos preiszugeben, damit mit einer auf diese Weise definierten Zielfunktion Nutzenwerte für jede Kombination von Software-Komponenten ermittelt werden können. Demgegenüber stützt sich der AHP auf eine Hierarchie von Zielen und errechnet die benötigten Gewichtungen durch paarweise Vergleiche aller Kriterien je Hierarchieebene (mit Hilfe der Eigenwertmethode). Damit können nun bei korrekter Anwendung – trotz vielfacher methodischer Kritik – tendenziell konsistentere Ergebnisse erzielt werden als mit der Nutzwertanalyse; dies muss allerdings mit enormem Aufwand für die in großer Zahl notwendigen paarweisen Vergleiche erkauft werden. Maiden illustriert das anhand einer Fallstudie zu einem Projekt mit 130 Anforderungen, bei dem die Verwendung des AHP daran scheiterte, dass die Entscheidungsträger mehr als 42.000 Vergleiche durchführen hätten müssen [Maiden1998].

Generell scheint jeder Versuch, die vielfältigen Aspekte der Komponentenauswahl bei der Software-Entwicklung auf einen einzigen „Nutzenwert“ zu reduzieren, problematisch, da durch die Aggregation verschiedener Ziele zu einem einzigen Indikator nicht zuletzt individuelle Attribute und somit die Information über besondere Schwächen oder Stärken verloren gehen.

Daher kann eine hohe Bewertung in einem Ziel eine schlechte Leistung in einem anderen Kriterium (gleichsam automatisch) ausgleichen und solcherart ein potentielles Risiko unentdeckt bleiben [Ncube2002]. Forscher wie Praktiker vertreten deshalb vielfach die Auffassung, dass traditionelle Methoden bei der Bewertung von Software-Komponenten ungeeignet sind [Martinsons1998,Ryan2004] und Entscheidungsträger bessere Unterstützung bei der Bewertung von Kombinationen von Software-Komponenten und der entsprechenden Verteilung von Ressourcen hinsichtlich der damit verbundenen Nutzen und Risiken benötigen.

3 Ein interaktives Entscheidungsunterstützungssystem

Der nachfolgend vorgestellte neue Ansatz möchte eine geeignete Alternative bieten. Er zeichnet sich insbesondere dadurch aus, dass Entscheidungsträger weder umfangreiche a priori Präferenzinformationen bereitstellen noch zahlreiche paarweise Vergleiche anstellen müssen. Stattdessen wird zuerst – ohne aktives Zutun der Entscheidungsträger – die Menge der Pareto-effizienten Kombinationen von Software-Komponenten bestimmt und im Anschluss die Möglichkeit geboten, darin interaktiv nach jener Lösung zu suchen, die den individuellen Präferenzen am besten entspricht. Vor einer detaillierten Diskussion der beiden Phasen des interaktiven Entscheidungsunterstützungssystems soll zunächst noch ein typisches Vorgehensmodell für die komponentenbasierte Software-Entwicklung skizziert werden, um zu zeigen, in welcher Phase unser Ansatz zum Einsatz kommen würde.

3.1 Das Vorgehensmodell OTSO

Das von Kontio vorgeschlagene Modell OTSO (Off-The-Shelf Option; [Kontio1995]) gliedert sich in die nachfolgend beschriebenen fünf Phasen.

1. Definition der Kriterien: Im ersten Schritt werden die relevanten Zielkriterien bestimmt mit besonderem Augenmerk auf die Infrastruktur der Organisation, die Applikationsarchitektur, das Applikationsdesign, Anforderungen an die Applikation, Projektziele und -einschränkungen und die grundsätzliche Verfügbarkeit von Software-Komponenten bzw. -bibliotheken. Die Auswahl der Kriterien muss für jedes Vorhaben separat (z.B. unter Verwendung von GQM [Basili1987]) durchgeführt werden, da etwa funktionale Anforderungen regelmäßig für jede Applikation unterschiedlich sind. Typische Kriterien gliedern sich grob in vier Kategorien, nämlich: (i) Funktionale Kriterien, die auf Basis der Geschäftsprozesse definiert werden und deren

Erfüllung üblicherweise ein besonders hoher Stellenwert zukommt; (ii) Qualitätskriterien, wie beispielsweise die Fehlerrate, Leistungsmaßzahlen, Benutzerfreundlichkeit oder Sicherheit; (iii) Strategische Kriterien mit einem Fokus auf Kosten, Zeitaspekten oder verfügbarem Personal und dessen Qualifikationen; und (iv) Einsatzspezifische und architekturrelevante Kriterien, die eine Rolle spielen, sofern dadurch Vorgaben oder Einschränkungen für das zu realisierende Softwareprodukt bedingt sind.

2. Identifikation der Komponenten: Dieser Schritt umfasst die Identifikation von potentiellen Komponenten, die beispielsweise untergliedert werden können in (i) Betriebssysteme (etwa AIX, FreeBSD, Linux, Microsoft Windows Server, QNX, Solaris), (ii) Middleware (BEA Weblogic, IBM Websphere, OpenORB, Oracle Application Server, Visibroker und weitere Komponenten wie z.B. CRM, DMS, ERP, SCM, SRM, die von unterschiedlichen Firmen angeboten werden), (iii) Datenbanken (Hypersonic SQL, Microsoft Access, Microsoft SQL Server, MySQL, Oracle DB, PostgreSQL) und (iv) Support Software (Agent++, Apache Tomcat, Hypertext Preprocessor, IBM Java Runtime Environment, Intel COPS Client, Log4J, Telia BER Coder, WebMacro, Xerces, XMLDB) [Morisio2002].

3. Screening: In dieser Phase sollen – zunächst einmal isoliert von den anderen – vielversprechende Komponenten für eine anschließende, umfassendere Evaluierung gefiltert werden.

4. Evaluierung: In der Evaluierungsphase erfolgt die Konsolidierung und gründliche Evaluierung der zuvor identifizierten Komponenten in Bezug auf die im ersten Schritt festgelegten Kriterien sowie die Dokumentation der Ergebnisse.

5. Analyse: In der Analysephase bestimmen die Entscheidungsträger die für die Umsetzung der Anforderungen „beste“ Kombination an Komponenten. Dazu wird in OTSO der AHP eingesetzt.

3.2 Die Bestimmung effizienter Kombinationen von Software-Komponenten

Unser Ansatz ist zum Einsatz in der letzten Phase des eben umrissenen Vorgehensmodells konzipiert und würde dort den althergebrachten AHP ersetzen. Der erste Schritt zielt dabei auf die Ermittlung der hinsichtlich der betrachteten Ziele Pareto-effizienten Kombinationen von Software-Komponenten. Dazu ist das zugrunde liegende multikriterielle kombinatorische Optimierungsproblem (zur multiobjective combinatorial optimization (MOCO) vgl. [Ehrgott2000]) zu lösen. Binäre Entscheidungsvariablen $x_i \in \{0,1\}$ geben darin an, ob eine Komponente *i* in einer Lösung verwendet wird oder nicht ($x_i = 1$ falls ja, ansonsten $x_i = 0$). Eine Kombination

von Komponenten wird durch einen Vektor $x = (x_1, ..., x_N)$ mit Einträgen für alle N zur Wahl stehenden Komponenten abgebildet. Im MOCO-Problem sind nun K Zielfunktionen $u_k(x)$ ($k = 1, ..., K$) (z.B. für die Funktionalität, Verwendbarkeit oder die Zuverlässigkeit) zu maximieren bzw. gegebenenfalls (etwa bei den Kosten) zu minimieren. Die Funktionen $u_k(x)$ können dabei beliebige Formen annehmen, solange sie für alle zulässigen Kombinationen von Software-Komponenten definiert sind. Dabei können auch vielfältige Abhängigkeiten zwischen zwei oder mehreren Komponenten (bedingt z.B. durch Synergie- oder Kannibalismuseffekte) berücksichtigt werden; vgl. [Stummer2003] für eine detaillierte Diskussion am Beispiel der Auswahl von F&E-Portfolios.

Jedweder Berechnungsalgorithmus, der in dieser ersten Phase zum Einsatz kommt, muss alle (oder zumindest nahezu alle) effizienten MOCO-Lösungen ermitteln und dabei insbesondere Nebenbedingungen aus zwei Gruppen berücksichtigen. Solche aus der ersten Gruppe betreffen typischerweise Ressourcenbeschränkungen (etwa bei den Entwicklungs- oder Wartungskosten). Sie haben daher in der Regel die Form $\sum_i r_{iq} x_i \leq R_q$ ($q = 1, ..., Q$), wobei r_{iq} für die Menge an von Komponente i nachgefragten Ressourcen vom Typ q steht und Parameter R_q das Gesamtbudget für diesen Ressourcentyp angibt. Im Falle von Synergie- oder Kannibalismuseffekten müssen wiederum entsprechende Korrekturterme ergänzt werden. Nebenbedingungen aus der zweiten Gruppe gewährleistet, dass jedenfalls eine Mindestzahl bzw. nicht mehr als eine Maximalanzahl von Komponenten aus einer vorab festgelegten Teilmenge ausgewählt werden. Zum Beispiel könnte vorgegeben werden, dass in jeder gültigen Lösung aus jeder Architekturschicht mindestens eine Komponente enthalten sein muss, falls der Benutzer etwa eine mehrschichtige Architektur abbilden möchte. Wenn das sechs Komponenten mit den Indizes 1 bis 6 betrifft, lautet die entsprechende Nebenbedingung: $\sum_{i=1}^{6} x_i \geq 1$.

Das MOCO-Problem ist NP-schwer, da sich der Suchraum mit jeder hinzukommenden Komponente verdoppelt. Abhängig vom Umfang – bestimmt insbesondere durch die Anzahl der zur Wahl stehenden Software-Komponenten, die Zahl an Zielen sowie Zahl und Typ der Abhängigkeiten zwischen den Komponenten – kann es entweder exakt durch vollständige Enumeration aller Kombinationen von Komponenten (bei 30 Komponenten wären das mehr als eine Milliarde ($2^{30} > 1{,}07 \cdot 10^9$) zu evaluierende Alternativen) oder mit Hilfe von (Meta-) Heuristiken gelöst werden. Als Daumenregel gilt, dass letztere ab etwa vierzig Komponenten

zum Einsatz kommen. Sie können zwar keine Garantie für das Auffinden aller Pareto-effizienten MOCO-Lösungen geben, ermitteln aber regelmäßig in einem Bruchteil der für die vollständige Enumeration benötigten Zeit bereits einen Großteil der gesuchten Lösungen; einen Überblick zu Metaheuristiken für MOCO-Probleme bietet [Ehrgott2004], Anwendungsbeispiele mit Leistungsvergleichen finden sich etwa bei [Doerner2006,Stummer2005]. Nichtsdestotrotz kann die vollständige Enumeration auch bei einer höheren Anzahl an Komponenten noch in praktikabler Zeit durchlaufen werden, sofern den Suchraum stark einschränkende Abhängigkeiten existieren (z.B. wenn aus jeder Architekturschicht exakt eine Alternative zu wählen ist).

3.3 Zwei Varianten für die interaktive Suche im Lösungsraums

In der zweiten Phase unterstützt das System den Entscheidungsträger bei der endgültigen Auswahl jener Kombination von Software-Komponenten, die am besten seinen Präferenzen entspricht. Angesichts eines Lösungsraums, der unter Umständen mehrere tausend Elemente umfasst, kommt eine simple Auflistung sicherlich nicht in Frage. Stattdessen bietet das System zwei alternative Varianten für die interaktive „Erforschung“ des Lösungsraums.

Die erste erlaubt es, sich durch Verschieben von unteren bzw. oberen Schranken für beliebige Ziele rasch im Lösungsraum zu bewegen. Das Entscheidungsunterstützende System (EUS) beginnt dazu mit der Anzeige von K „fliegenden“ Säulen (vgl. Abbildung 1).

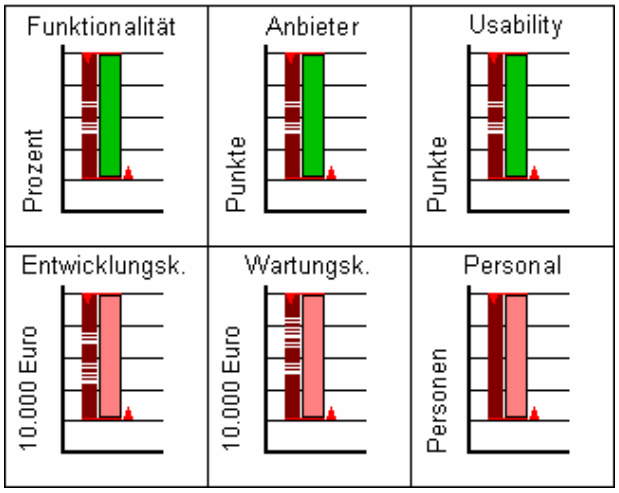

Abbildung 1: Status des EUS zu Beginn der interaktiven Auswahl

Für jedes Ziel (vgl. Abbildung 2) bietet das System Informationen über (i) die Zielwerte, die mit zumindest einer der effizienten Lösungen machbar wären (die kurzen Markierungen auf der linken Seite können dazu optisch zu schmalen Säulen zusammenwachsen, wenn sie knapp bei-

einander liegen) sowie (ii) den Zielgrößenbereich, der mit den – unter Bedachtnahme auf die bereits gesetzten Ober- bzw. Untergrenzen – noch verbliebenen Lösungen erreicht werden kann.

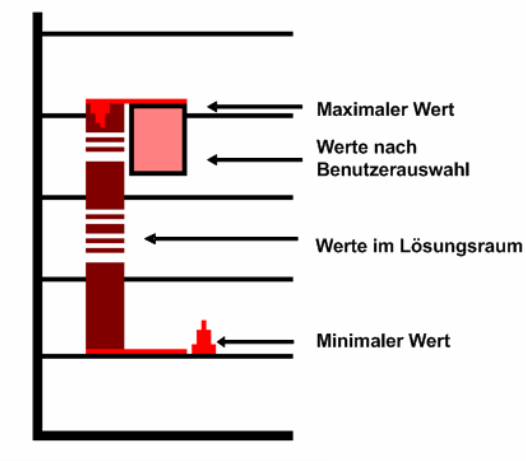

Abbildung 2: Detaillierte Darstellung

Die beweglichen, horizontalen Linien mit den kleinen Pfeilen auf einer Seite symbolisieren die oberen und unteren Schranken und erlauben die schrittweise Einschränkung (z.B. durch Anheben der unteren Schranke in einem Ziel) oder Ausweiterung (z.B. durch Hinaufsetzen einer oberen Schranke) der Menge jener Pareto-effizienten MOCO-Lösungen, die gegen keine der bislang gesetzten Schranken verstoßen. Damit kann der Entscheidungsträger spielerisch Vorgaben machen und erhält vom System unmittelbare Rückmeldung über deren Auswirkungen. So illustriert etwa Abbildung 3 die Rückmeldung, nachdem die Obergrenze für die Entwicklungskosten gesenkt worden ist.

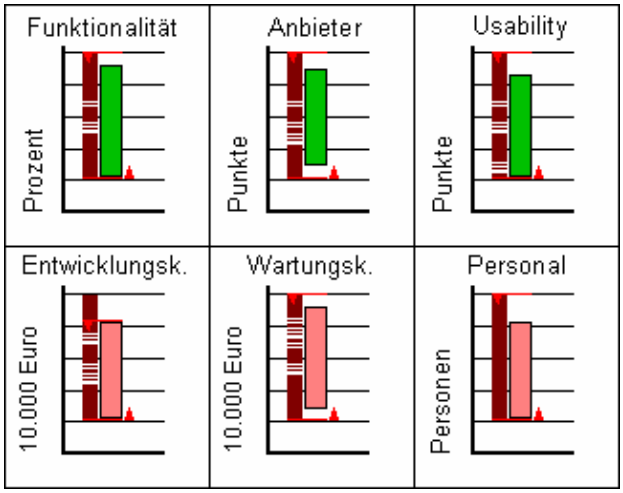

Abbildung 3: Status des DSS nach einer ersten Einschränkung

Dadurch scheiden nämlich jene Lösungsalternativen mit besonders hohen Entwicklungskosten (häufig auch korreliert mit höherem Personalbedarf), jedoch oftmals auch niedrigeren Wartungskosten bzw. höherer Funktionalität aus. Dementsprechend schrumpfen die, durch die fliegenden Balken repräsentierten, Intervalle der nunmehr noch in den anderen Zielen erreichbaren Werte. Eine anschließende Erhöhung der unteren Schranke in der Kategorie „Funktionalität“ führt zu einer weiteren Reduktion der Alternativen auf jene, die beide Vorgaben (d.h. reduzierte Entwicklungskosten und höhere Mindest-Funktionalität) erfüllen, wobei hierbei insbesondere kostengünstigere Alternativen wegfallen (vgl. Abbildung 4).

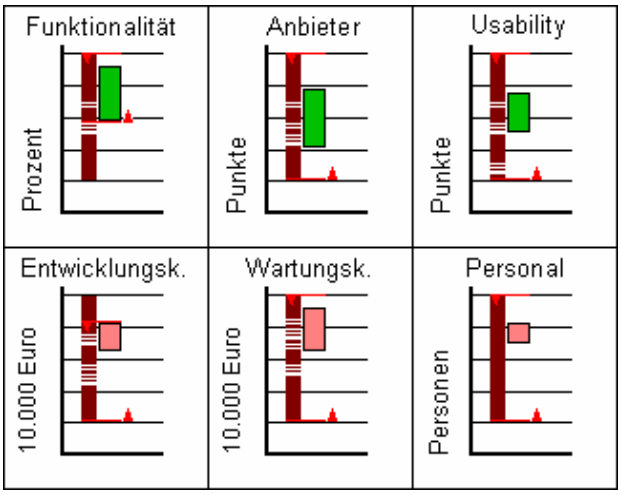

Abbildung 4: Status des DSS nach einer zweiten Einschränkung

In weiteren Iterationen erlaubt das System dem Entscheidungsträger die spielerische Modifikation beliebig vieler oberer und unterer Schranken und ermöglicht ihm damit, unmittelbar die Konsequenzen seiner Vorgaben zu erfahren. Er gewinnt auf diese Weise ein besseres „Gefühl“ für das Entscheidungsproblem und sollte schließlich auf jene Lösung stoßen, die für ihn den „besten“ Mix an Zielwerten bietet und bei der er nicht mehr bereit ist, weitere Kompromisse zwischen den Zielen einzugehen. Dabei hat er – im Gegensatz zu herkömmlichen Verfahren – weder explizit Gewichtungen für einzelne Ziele anzugeben noch muss er tausende paarweise Vergleiche über sich ergehen lassen. Stattdessen erhält er umfassende Informationen über das Auswahlproblem und kann sich darauf verlassen, dass jede angebotene Lösung effizient ist und somit keine Alternative existiert, die objektiv „besser“ wäre.

Im zweiten von uns vorgeschlagene Ansatz für die interaktive Suche im Lösungsraum präsentiert das EUS dem Entscheidungsträger in jeder von mehreren Runden die Zielwerte für bis zu sieben Kombinationen von Software-Komponenten, aus denen die jeweils attraktivste zu wählen ist (vgl. Abbildung 5 für ein Beispiel mit vier Lösungen). Die Beschränkung auf maxi-

mal sieben Alternativen ist psychologisch begründet, weil Menschen sieben plus/minus zwei Elemente gerade noch gut vergleichen können.

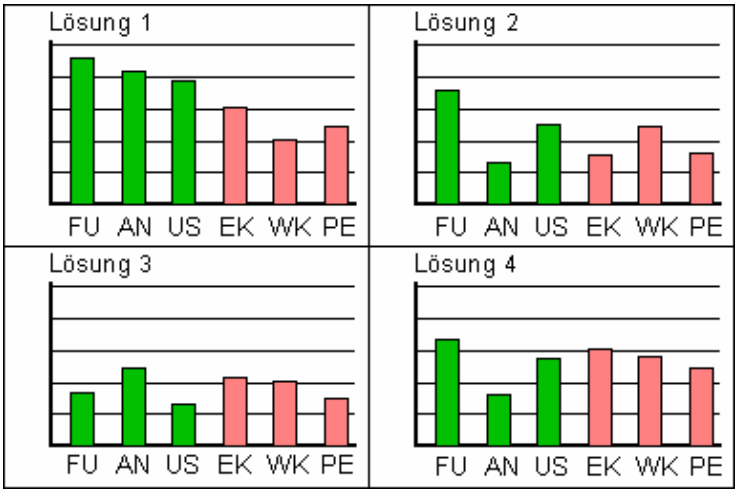

Abbildung 5: Auswahl mit Hilfe von Cluster-Repräsentanten

Anfangs unterscheiden sich die Zielwerte der Lösungsvorschläge in hohem Maße, so dass hier vor allem Richtungsentscheidungen (etwa hohe Funktionalität vs. geringe Kosten) zu treffen sind. Später wird die Suche verfeinert und die Unterschiede zwischen den angebotenen Lösungen werden zusehends geringer. Methodisch kommt das k-means-Clustering zum Einsatz. Aus den damit aus der Gesamtmenge aller effizienten Lösungen bestimmten sieben Clustern werden dem Entscheidungsträger Repräsentanten angeboten. Jenes Cluster, aus dem der dann gewählte Vertreter stammt, wird in der nächsten Iteration weiter in Sub-Cluster unterteilt usw., bis sich der Entscheidungsträger letztlich für ein Cluster entscheidet, das nur mehr eine einzige Lösung enthält. Bildlich gesprochen „taucht“ er dabei gleichsam in den Lösungsraum ein und bewegt sich zuerst in großen, dann später immer kleineren Kreisen an die für ihn individuell attraktivste Lösung heran. Dies funktioniert in aller Regel recht rasch, weil im Durchschnitt jedes Cluster nur ein Siebentel der noch verbliebenen Lösungen enthält. Selbst bei etwa 20.000 effizienten Lösungen sind daher im Mittel lediglich um die fünf, mit hoher Wahrscheinlichkeit aber jedenfalls nicht mehr als zehn Iterationen nötig.

4 Resümee und Ausblick

Entscheidungsträger sehen sich bei der komponentenbasierten Software-Entwicklung regelmäßig mit der Herausforderung konfrontiert, aus einem großen Pool von Komponenten die

„beste“ Kombination zusammen zu stellen. Diese Aufgabe wird nicht nur durch eine oftmals große Zahl an alternativen Komponenten (auf verschiedenen architektonischen Ebenen), sondern auch durch Abhängigkeiten zwischen einzelnen Komponenten bzw. nicht zuletzt durch mehrere divergierende Zielsetzungen verkompliziert. In diesem Aufsatz haben wir einen zweiphasigen Ansatz zur Entscheidungsunterstützung bei der Auswahl von Software-Komponenten entwickelt, der als Erweiterung in bestehende Vorgehensmodelle für die komponentenbasierte Softwareentwicklung integriert werden kann und insbesondere Probleme und Unzulänglichkeiten, die bei herkömmlichen Verfahren wie der Nutzwertanalyse oder dem AHP auftreten, überwindet.

Wie alle anderen Verfahren hängt auch unseres von der Verfügbarkeit geeigneter Zielfunktionen bzw. der Qualität der in der Evaluierungsphase erhobenen Daten ab. So ist es sicherlich nicht trivial, eine geeignete Funktion für die Zuverlässigkeit einer Kombination von Software-Komponenten aufzustellen. Damit wollen wir uns im Zuge weiterer Forschungsaktivitäten beschäftigen, wobei wir hierfür etwa das Einbeziehen der unmittelbar betroffenen Praktiker im Rahmen von Workshops ins Auge gefasst haben (für ein Beispiel vgl. [Neubauer2006]).

Darüber hinaus gibt es eine Reihe weiterer interessanter Fragestellungen. So sollten Aspekte der Ungewissheit berücksichtigt werden, um bei Bedarf bewusst robuste Zusammenstellungen von Software-Komponenten erzwingen zu können, die dann weniger anfällig für sich ändernde Systemanforderungen wären. Im MOCO-Modell ließe sich das in einem ersten Schritt mit stochastischen Zufallsvariablen und dann über Quantilswerte als separate Ziele umsetzen (für ein Beispiel vgl. [Medaglia2006]). Des Weiteren könnten vermehrt Ideen aus der Literatur zu Gruppenentscheidungen bzw. der Verhandlungsanalyse übernommen werden. Und schließlich sollen Feldstudien Aufschluss über die Eignung unterschiedlicher (grafischer) Benutzerschnittstellen geben.

Literaturverzeichnis

[Alves2001] C. Alves, J. Castro (2001) CRE: A systematic method for COTS components selection. Proceedings of the XV Brazilian Symposium on Software Engineering.

[Alves2003] C. Alves, A. Finkelstein (2003) Investigating conflicts in COTS decisionmaking. International Journal of Software Engineering and Knowledge Engineering, 13. Jg., H. 5, S. 473-495.

[Andrews2005] A. A. Andrews, A. Stefik, N. Picone, S. Ghosh (2005) A COTS component comprehension process. Proceedings of the 13th International Workshop on Program Comprehension (IWPC'05), IEEE, S. 135-144.

[Basili1987] V. R. Basili, H. Rombach (1987) Tailoring the software process to project goals and environments. Proceedings of the 9th International Conference on Software Engineering. IEEE, S. 345-357.

[Basili1996] V. R. Basili, L. C. Briand, W. L. Melo (1996) How reuse influences productivity in object-oriented systems. Communications of the ACM, 39. Jg., H. 10, S. 104-116.

[Doerner2006] K. F. Doerner, W. J. Gutjahr, R. F. Hartl, C. Strauss, C. Stummer (2006) Nature-inspired metaheuristics for multiobjective activity crashing. Omega, im Druck.

[Ehrgott2000] M. Ehrgott, X. Gandibleux (2000) A survey and annotated bibliography of multiobjective combinatorial optimization. OR Spectrum, 22. Jg., H. 4, S. 425-460.

[Ehrgott2004] M. Ehrgott, X. Gandibleux (2004) Approximative solution methods for multi-objective combinatorial optimization. Top, 12. Jg., H. 1, S. 1-63.

[Kontio1995] J. Kontio (1995) OTSO: A systematic process for reusable software component selection. Institute for Advanced Computer Studies and Department of Computer Science, University of Maryland, Arbeitspapier.

[Kunda2003] D. Kunda (2003) STACE: Social technical approach to COTS software evaluation. Component-Based Software Quality: Methods and Techniques, Springer LNCS 2693, S. 64-84.

[LozanoTello2002] A. Lozano-Tello, A. Gomez-Perez (2002) BAREMO: How to choose the appropriate software component using the Analytic Hierarchy Process. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering, ACM Proceedings 27, S. 781-788.

[Maiden1998] N. Maiden, C. Ncube (1998) Acquiring COTS software selection requirements. IEEE Software, 15. Jg., H. 2, S. 46-56.

[Maiden2002] N. Maiden, H. Kim, C. Ncube (2002) Rethinking process guidance for selecting software components. Proceedings of the First International Conference on COTS-Based Software Systems (ICCBSS 2002), Springer LNCS 2255, S. 151-164.

[Martinsons1998] M. Martinsons, R. Davidson, D. Tse (1998) The balanced scorecard: A foundation for the strategic management of information systems. Decision Support Systems Journal, 25. Jg., H. 1, S. 71-78.

[Medaglia2006] A. Medaglia, S. Graves, J. Ringuest (2006) A multiobjective evolutionary approach for linearly constrained project selection under uncertainty. European Journal of Operational Research, im Druck.

[Morisio2002] M. Morisio, M. Torchiano (2002) Definition and classification of COTS: A proposal. Proceedings of the First International Conference on COTS-Based Software Systems (ICCBSS 2002). Springer LNCS 2255, S. 165-175.

[Navarrete2005] F. Navarrete, P. Botella, X. Franch (2005) How agile COTS selection methods are (and can be)? Proceedings of the 31st EUROMICROConference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'05), IEEE, S. 160-167.

[Ncube2002] C. Ncube, J. C. Dean (2002) The limitations of current decision-making techniques in the procurement of COTS software components. Proceedings of the First International Conference on COTS-Based Software Systems (ICCBSS 2002), Springer LNCS 2255, S. 176-187.

[Neubauer2006] T. Neubauer, C. Stummer, E. Weippl (2006) Workshop-based multiobjective security safeguard selection. Proceedings of the First International Conference on Availability, Reliability and Security (ARES'06). IEEE, S. 366-373.

[Ruhe2002] G. Ruhe (2002) Intelligent support for selection of COTS products. Revised Papers from the Workshop on Web, Web-Services, and Database Systems (NODe 2002), Springer LNCS 2593, S. 34-45.

[Ruhe2003] G. Ruhe (2003) Software engineering decision support: A new paradigm for learning software organizations. Proceedings of the 4th International Workshop on Advances in Learning Software Organizations (LSO 2002), Springer LNCS 2640, S. 104-113.

[Ryan2004] S. D. Ryan, M. S. Gates (2004) Inclusion of social sub-system issues in IT-investment decisions: An empiricial assessment. Information Resources Management Journal, 17. Jg., H. 1, S. 1-18.

[Torchiano2004] M. Torchiano, M. Morisio (2004) Overlooked aspects of COTS-based development. IEEE Software, 21. Jg., H. 2, S. 88-93.

[Tran1997] V. Tran, D.-B. Liu, B. Hummel (1997) Component-based systems development: challenges and lessons learned. Software Technology and Engineering Practice. IEEE, S. 452-462.

[Stummer2003] C. Stummer, K. Heidenberger (2003) Interactive R&D portfolio analysis with project interdependencies and time profiles of multiple objectives. IEEE Transactions on Engineering Management, 50. Jg., H. 2, S. 175-183.

[Stummer2005] C. Stummer, M. Sun (2005) New multiobjective metaheuristic solution procedures for capital investment planning. Journal of Heuristics, 11. Jg., H. 3, S. 183-199.

[Wanyama2005] T. Wanyama, B. Homayoun (2005) Towards providing decision support for COTS selection. Proceedings of the 18th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2005), IEEE, S. 908-911.