Software aus "Standardkomponenten" - eine Quelle neuer Probleme ?

Peter F. Elzer
Institut für Prozeß- und Produktionsleittechnik (IPP) der TU Clausthal (TUC)
Julius-Albert-Straße 6, 38678 Clausthal-Zellerfeld, e-mail: elzer@ipp.tu-clausthal.de

Zusammenfassung

Im Folgenden wird ein Aspekt der Softwareentwicklung dargestellt, der bisher in der Diskussion wenig berücksichtigt wurde: der Zusammenbau von Anwendungssoftwaresystemen aus Standardkomponenten. Es wird versucht, einige der dabei entstehenden Probleme darzustellen, mögliche Gründe dafür zu identifizieren und einige erfolgversprechende Abhilfemaßnahmen zu diskutieren. Um eine Materialbasis für eine vertiefte Diskussion aufzubauen, ist im Anhang ein Fragebogen beigefügt.

1 Einleitung

Die "klassische" Neuentwicklung von Software findet inzwischen auf vielen Anwendungsgebieten nicht mehr statt. Heute werden häufig "Standardpakete" eingesetzt oder - wie etwa in der Automatisierungstechnik oder bei Anwendungen im Bereich der "neuen Medien" - Anwendungssoftware wird aus auf dem Markt verfügbaren "Standardkomponenten" zusammengesetzt. Für diese Tendenz gibt es gute Gründe. Sie wirft aber viele neue Probleme bei der Anwendungsentwicklung auf, die deshalb in Zukunft berücksichtigt werden müssen. Sie wurden aber bisher nur in geringem Umfang thematisiert [El97, Lö02, EL02], da sie nicht in allen Branchen der Softwareentwicklung auftreten oder noch nicht genügend Fakten gesammelt wurden.

Der Verfasser möchte deshalb - wie auf der gemeinsamen Sitzung der GI-Fachgruppe 5.1.2 (Projektmanagement) und des Arbeitskreises "Management von Softwareprojekten" in der Fachgruppe 2.1.1 (Softwaretechnik) am 22.05.2003 in Dortmund besprochen - diese Darstellung nutzen, um Mitglieder der Fachgruppen und des AK zu bitten, ihm Argumente, Zahlen und Fakten aus ihrem Erfahrungsbereich mitzuteilen, die zu einer weiteren Untersuchung der Problematik dienlich sein könnten.

Aus diesem Grund wird der Artikel auch in geringfügig erweiterter Form [El04] im WI-MAW-Rundbrief, dem Organ des GI-Fachausschusses "Management der Anwendungsentwicklung und -wartung" erscheinen. Als kleiner Leitfaden für die Datensammlung ist der Fragebogen im Anhang gedacht. Er soll aber nicht als vollständig und endgültig betrachtet werden. Im Gegenteil, Vorschläge für Änderungen oder Erweiterungen sind äußerst willkommen.

2 Beispiele

Zunächst sollen zwei Beispiele dargestellt werden, um die Überlegungen etwas weniger abstrakt - oder vielleicht sogar "fremdartig" - erscheinen zu lassen.

2.1 Leitsysteme in der Automatisierungstechnik

2.1.1 Situationsbeschreibung

Software für Anwendungen in der Automatisierungstechnik wurde seit Mitte der 60er bis Anfang der 80er Jahre meist von "Systemfirmen" in Form konfigurierbarer Pakete für bestimmte Anwendungsgebiete erstellt, wie z.B. Versorgungsnetze, Kraftwerke, Chemieanlagen, Fertigungseinrichtungen. Daraus wurde die Software für die einzelnen Kunden entsprechend deren jeweiligen konkreten Anforderungen abgeleitet. Dabei kamen verschiedene "Generierungstechniken" zum Einsatz.

Die Software stützte sich jeweils auf eine bestimmte Rechnerfamilie mit mitgeliefertem Betriebssystem und anderer Hilfssoftware (z.B. Compiler, math. Laufzeitpaket, Treiber für die Standardperipherie), die üblicherweise ebenfalls mitgeliefert wurden. Die Rechner waren spezielle "Prozeßrechner" von einschlägigen Firmen (z.B. DEC, Siemens, HP, Honeywell, Ferranti). Spezielle Peripheriegeräte für die Automatisierungstechnik (z.B. Meßfühler, Fernwirktechnik, etc.) und die sehr ausgefeilten Bedienschnittstellen wurden meist vom Hersteller des Gesamtsystems gebaut. Treiber und Grafiksoftware wurden ebenfalls als Bestandteile der Anwendungssoftware entwickelt. Das auszuliefernde System bestand also im Wesentlichen aus den in Tabelle 1 aufgeführten Teilen.

		Lieferant:	
1	Anwendungs-SW	Systemfirma	
2	math. LZP		Rechnerfirma
3	Treiber	Systemfirma	Rechnerfirma
4	BS		Rechnerfirma
5	Rechner HW		Rechnerfirma
6	Peripherie HW	Systemfirma	Rechnerfirma

Tab.1: Lieferantenstruktur bei "alter Technologie"

Die Vertragsbeziehungen waren relativ klar und einfach: Die Systemfirma trug die Gesamtverantwortung und hatte auch für alle Komponenten Garantie zu leisten. Mit der Rechnerfirma bestand üblicherweise ein OEM-Vertrag, der die Systemfirma verpflichtete, ausschließlich die Hardware der Rechnerfirma einzusetzen. Diese hatte die Systemfirma rechtzeitig über geplante Produktänderungen zu informieren und über einen sinnvollen Zeitraum Ersatzteile vorzuhalten. Natürlich funktionierte das nicht immer völlig reibungslos; die Verhältnisse waren aber überschaubar. Gegen Mitte der 80er Jahre änderte sich die Situation jedoch dramatisch:

1 Viele Kunden verlangten den Einsatz "preisgünstiger" Arbeitsplatzrechner ("PC's") anstelle der (relativ) teu-

ren Prozeßrechner.

2 Durch die Verfügbarkeit leistungsfähiger Grafik auch bei Bürorechnern vergrößerte sich der Markt für hochentwickelte Grafiksoftware plötzlich um Größenordnungen. Er wurde von entsprechenden Herstellern schnell erkannt und ausgefüllt. Die Preise für Grafiksoftware sanken so rapide, daß sie nicht mehr im Rahmen der automatisierungstechnischen Anwendungssoftware entwickelt werden konnte, sondern zugekauft werden mußte.

3 Die Preise für die "PC's" verfielen so rasch, daß viele Hersteller meist nicht lange genug überlebten, um zuverlässige Partner für OEM-Verträge zu sein. Systemhersteller müssen sich inzwischen fast bei jedem neuen Projekt neu für eine Hardwarebasis entscheiden.

4 Diese Situation macht es kaum mehr möglich, für automatisierungstechnische Anwendungen geeignete Betriebssysteme zu entwickeln und anzubieten. So müssen in den meisten Fällen Produkte eines sehr bekannten Herstellers eingesetzt werden, die aber wiederum für einen viel zu breiten Kundenkreis bestimmt sind, um die besonderen Anforderungen der Automatisierungstechnik optimal erfüllen zu können.

5 Auch Compiler und mathematische Laufzeitpakete werden nicht mehr vom Rechnerhersteller geliefert. Damit ist nicht mehr gewährleistet, daß sie auf den verwendeten Rechnertyp oder die aktuelle Betriebssystemsversion optimal abgestimmt sind. Für Anwendungen mit einem hohen Gehalt an komplexer Regelungstechnik werden ebenfalls spezielle Laufzeitpakete eingesetzt, die wieder von einem anderen Lieferanten stammen.

6 Auch in Bezug auf die Datenhaltung hat sich die Situation geändert. Früher wurden die in einer Anwendungsklasse üblichen Dateiformate und -strukturen von der Systemfirma auf der Basis ihres Spezialwissens selbst konzipiert und realisiert. Heute werden weitgehend allgemein verwendbare Datenbanken eingesetzt, die von darauf spezialisierten Firmen angeboten werden.

Tabelle 2 illustriert ein Beispiel für die neue Situation, bei dem mindestens sieben (!) - statt vorher zwei - Partner zu koordinieren sind.

Für Menschen mit Managementerfahrung erscheint dies zunächst nicht als ein wesentliches Problem. Eine "ARGE" in der Bauindustrie hat oft ähnlich viele Partner und ein Automobilhersteller mehr Zulieferer. Eine ARGE ist aber nur ein Zweckbündnis auf Zeit, das üblicherweise mit Fertigstellung des Bauvorhabens endet. Die Zulieferer stehen in einem strengen Abhängigkeitsverhältnis zum Automobilwerk, das größer und finanzkräftiger ist.

Wie in [El97] dargelegt, ist in der Automatisierungstechnik die Situation anders. Die "Systemfirma" ist entweder ein "mittelständisches" Systemhaus oder ein Geschäftsbereich einer großen Firma. Die Komponentenlieferanten (speziell Fremdfirmen 1, 3 und 5) sind häufig Weltkonzerne mit Monopolcharakter. Die Zusammenarbeit kann aber nicht nach Fertigstellung eines Automatisierungssystems aufgegeben werden, da die Systemfirma für einen angemessenen RoI ihres Eigenanteils mehrere Kunden benötigt. Außerdem muß sie für das Gesamtsystem in der Regel länger Garantie leisten als die Lebensdauer eines üblichen "PC" beträgt. Ihr Hauptgeschäft besteht auch meist nicht in der Erstellung und Lieferung von Software, sondern in der Herstellung und dem Verkauf wesentlicher Teile der eigentlichen technischen Anlage sowie in Planung und Engineering des Systems.

		Lieferant:	
1	Grafikpaket		Fremdfirma 1
2	Anwendung	Systemfirma	
3	math. LZP		Fremdfirma 2
4	Datenbank		Fremdfirma 3
5	Treiber	Systemfirma	Fremdfirma 4
6	Betriebssystem		Fremdfirma 5
7	Rechner HW		Fremdfirma 6
8	Peripherie HW	Systemfirma	Fremdfirma 4

Tab.2: Lieferantenstruktur bei "neuer Technologie"

2.1.2 Analyse

2.1.2.1 Verringerung der Entwicklungskosten

Zunächst ist festzustellen, daß die Situation unumkehrbar ist, so unbefriedigend sich die derzeitige Lage auch darstellen mag. Das ergibt sich klar aus einem ganz groben - und oberflächlichen - Vergleich der Entwicklungskosten:

Bei der Entwicklung eines Netzleitsystems "alter Bauart", betrug z.B. der Umfang der von der Systemfirma entwickelten Anwendungssoftware etwa 800 kLSC (Tab.3). Nimmt man für die Produktivität einen (seit vielen Jahren stabilen) Richtwert von 3 kLSC/MJ an, so ergibt sich ein Entwicklungsaufwand von 267 MJ. Das entspricht in etwa einem Team von 50 Mitarbeitern über einen Zeitraum von 5 Jahren.

	alt	neu
Anwendung	≈ 800 (kLSC)	≈ 200
UIMS	in Anwendung enthalten	> 1.000
LZP	≈ 50	> 100
DBMS	in Anwendung enthalten	> 1.000
Treiber	≈ 50	> 100
BS	≈ 100	> 1.000
Gesamt	≈ 1.000	> 3.400

Tab.3: Größenvergleich von Leitsystemen "alter" und "neuer" Technologie

Ein vergleichbares System "neuer Bauart" enthält (nach einer ebenfalls in Tab. 3 dargestellten Schätzung) nur noch etwa 200 kLSC (ein Viertel) Eigenentwicklungsanteil. Daraus ergibt sich (bei angenommener konstanter Produktivität) ein Entwicklungsaufwand von ca. 67 MJ, was einem Team von 22 Mitarbeitern über 3 Jahre entspricht. Damit wird das Projekt - aus Sicht der Entwicklung - nicht nur billiger, sondern erscheint zunächst auch viel leichter beherrschbar.

Dennoch wurde - nach Aussage Beteiligter - der Entwicklungsprozeß insgesamt schwieriger. Dazu kommt, daß Systeme neuer Bauart fehleranfälliger sind als dies früher üblich war. Bei genauerer Betrachtung lassen sich leider plausible Gründe dafür angeben.

2.1.2.2 Erhöhung der Komplexität

Zunächst wirkt die Zunahme der Zahl der getrennt zu betrachtenden Komponenten sehr moderat: das "neue" System besteht aus 8 gegenüber 6 Komponenten beim "alten". Betrachtet man aber die **Zahl der Schnittstellen**, so ergibt sich ein anderes Bild: Sie folgt dem "Brooks' schen Gesetz" [Br75]: **N = n(n-1)/2** ! (N = Anzahl der Schnittstellen, n der Komponenten).

In der "alten" Situation gab es 2 Lieferanten und 6 Komponenten. Daraus ergeben sich 15 technische und eine organisatorische Schnittstelle. Bei der "neuen" Technologie mit 8 Komponenten und mindestens 7 Partnern errechnen sich dagegen 28 technische und 21 (!) organisatorische Schnittstellen. Weiterhin kann eine Änderung in einer Komponente im Extremfall Änderungen an den Schnittstellen zu allen anderen Komponenten nach sich ziehen.

2.1.2.3 Erhöhung der Änderungsgeschwindigkeit

Es ist allgemein bekannt, daß Menschen eine Situation als besonders unangenehm empfinden, wenn sie den Eindruck eines "Kontrollverlustes" bekommen, z.B. durch einen von außen aufgezwungenen Zeittakt, dem sie sich nicht gewachsen fühlen. Hier zeigen sich dramatische Unterschiede zwischen "alter" und "neuer" Situation:

Nach einer alten Faustregel kann man für den Abstand zwischen Rechner- oder Software"generationen" ("Releases") ca. 18 Monate annehmen. In der "alten" Situation hatte man es mit einem Geschäftspartner zu tun, konnte sich also auf die Änderungen einstellen.

Heute muß man (mindestens) 6 Lieferanten berücksichtigen, was zu einem "Änderungstakt" von (im Mittel) 3 Monaten führt. Das erweckt bei vielen Menschen den Eindruck einer "zunehmenden Geschwindigkeit des technischen Fortschritts", ist aber nur der Verlust der Koordinierung der technischen Weiterentwicklung, deren Geschwindigkeit selbst sich nicht geändert hat. Ungeachtet dessen ist die Wirkung dieses Effekts auf die Projektmitarbeiter verheerend.

2.1.2.4 Erhöhte Fehleranfälligkeit

Als wesentliches Argument für den Ersatz von Eigenentwicklungen durch "Standardkomponenten" wird oft angeführt, daß die letzteren durch ihren breiteren Einsatz besser ausgetestet seien als noch so sorgfältig entwickelte Spezialsoftware. Man kann z.B. annehmen, daß eine sehr sorgfältig getestete Eigenentwicklung einen "Austestungsgrad" von 90% habe, eine Standardkomponente einen von 95%. Nach einer alten Erfahrungsregel kann man weiter davon ausgehen, daß ungetesteter Code 3 bis 20 Fehler pro 1.000 LOC enthält. Es verbleiben also im selbst entwickelten Code 0,3 bis 2 Fehler pro kLOC und in zugekauftem 0,15 bis 1.

Wendet man die Zahlen jedoch auf ein konkretes Beispiel an, so ergeben sich überraschende Ergebnisse. Zu diesem Zweck wurden in Tabelle 3 die Größe eines Leitsystems "alter Technologie" (an die sich der Verfasser erinnert) und die (geschätzte) eines "neuen" Systems gegenübergestellt. Letztere wurde nach Meinung des Verfassers vielleicht sogar noch als etwas zu klein angenommen. Das kann nur bedeuten, daß das in Tabelle 4 dargestellte Fehlerproblem noch drückender ist.

	Größe alt (kLOC)	Anzahl Fehler	Größe neu	Anzahl Fehler
Eigen-entwick-lung	≈ 800	240 bis 1.600	≈ 200	60 bis 400
Fremd-software	≈ 200	30 bis 200	> 3.200	480 bis 3.200
Gesamt	≈ 1.000	**270 bis 1.800**	> 3.400	**540 bis 3.600**

Tab.4: Vergleich der Anzahl verbleibender Fehler

Diese Betrachtung erklärt eine häufig gemachte Beobachtung: die Standzeiten neuerer Softwaresysteme sind geringer als die älterer!

Natürlich handelt es sich hier um absolute "worstcase" Abschätzungen. So sind z.B. seit langem Verfahren zur Minimierung der Anzahl von Schnittstellen in einem System bekannt. Nichtsdestotrotz werden dadurch die in der Praxis "gefühlten" Probleme beim Management der Entwicklung und der Pflege von "Komponentensoftware" plausibel.

2.2 Lehrsystem

Das zweite Beispiel kann aus Platzgründen nicht im Detail dargestellt werden. Es handelt sich um ein webbasiertes multimediales Lehr- und Lernsystem, das am IPP der TUC entwickelt wurde und seit 1998 im regulären Lehrbetrieb eingesetzt wird [LE99, LE00]. Es enthält neben dem selbst entwickelten Anwendungssystem Komponenten von 10 verschiedenen Lieferfirmen.

Diese Lieferantenvielfalt war technisch notwendig und ist nicht das wesentliche Problem. Die Schwierigkeiten bei der Pflege der Lehrinhalte ("Contents") beruhen ausschließlich auf dem Marktaustritt einer Komponente - des ursprünglich verwendeten Grafikpakets. Dadurch ist es kaum mehr möglich, die im Verlauf von etwa 12 Jahren entwickelten Lehrinhalte im Gegenwert von mehreren tausend Mannstunden mit einem Aufwand zu pflegen und weiterzuentwickeln, der für ein kleines Institut tragbar ist. Eine Neuerstellung ist aber völlig unmöglich.

Es müssen deshalb zur Zeit noch - unter entsprechenden Vorsichtsmaßnahmen - einige alte Rechner in Betrieb gehalten werden, auf denen das ursprüngliche Grafikpaket lauffähig ist ("Methode Rechnermuseum").

Damit wird an einem klein aussehenden Beispiel ein

Problem deutlich, das für die gesamte Branche, die sich mit "neuen Medien" beschäftigt, existenzbedrohend werden kann: der Verlust der gesamten Investitionen in "Contents", wenn ein Hersteller von Standardsoftware aus dem Markt austritt oder wenn er sein Dateiformat ändert, ohne die Aufwärtskompatibilität zu garantieren. Diese Investitionen können ein Vielfaches des Aufwands für die Systementwicklung oder - vor allem - des Preises der entsprechenden Komponente ausmachen.

3 Konsequenzen und Lösungsansätze

3.1 Identifikation von Einflußgrößen

So wirkt z.B. die **Anzahl der inneren Schnittstellen** in einem System aus Standardkomponenten nach der "Brooks'schen Formel" [N = n*(n - 1)/2] als ein sehr plausibles Maß. Bei näherer Betrachtung scheint die Analogie zur Entwicklung in einem Team tragfähig. Man kauft ja nicht Komponenten und faßt sie dann nicht mehr an, sondern kooperiert mit den Komponentenlieferanten über längere Zeit. Also geht deren Meinungsbildungsprozeß ähnlich in den Entwicklungsprozeß ein wie der von Teammitgliedern bei den Betrachtungen von Brooks.

Schwieriger ist schon das Problem der Fehleranfälligkeit der Systeme durch überflüssige Größe der zu verwendenden Standardkomponenten. Der Verfasser hat dafür keinen Lösungsvorschlag. Es scheint so, daß die Marktmacht eines Großlieferanten bei den anderen Beteiligten eine Art Lähmung hervorruft, wie sie das berühmte Kaninchen vor der Schlange befällt.

3.2 Modifikationen des Lebensdauerzyklus

Ursprünglich hatte der Verfasser angenommen, daß die Entwicklung von Softwaresystemen aus Standardkomponenten eine neue Strukturierung des Lebensdauerzyklus zur Folge haben müßte. Inzwischen ist aber ein anderer Eindruck entstanden: die Phasen bleiben dieselben, es ändert sich nur die Kostenverteilung. Für die Abschätzung von Entwicklungs- und Folgekosten ist aber von Interesse, wo man Zunahmen, wo Abnahmen erwarten würde und wo sie tatsächlich auftreten. In Tabelle 5 wurde versucht, die erwarteten Änderungen darzustellen. Nähere Begründungen existieren, würden aber den Rahmen dieses Papieres sprengen.

Eine erste Umfrage bei Mitarbeitern des IPP, die Softwaresysteme aus Standardkomponenten erstellt hatten, ergab aber andere Ergebnisse. Besonders fiel die Abweichung des Wertes für Integration und Systemtest von den Erwartungen auf. Natürlich ist die Zahlenbasis noch etwas zu schmal, um Schlüsse zu erlauben, doch erscheint die Diskrepanz sehr diskussionswürdig.

Ein ernstes Problem scheinen die "indirekten Änderungszwänge" bei der Wartung zu sein: man muß z.B. eine Komponente durch eine neue Version ersetzen, die aber eine neue Version des Betriebssystems braucht, was dazu führt, daß man alle anderen Komponenten durchsehen muß, ob diese wiederum an die neue BS-Version angepaßt werden müssen.

Auch der Aufwand für Analyse und Auswahl der zugekauften Komponenten sowie für das Erlernen ihrer Funktionalität wurde bisher kaum quantitativ erfaßt. Dem entspricht bei der Kostenverteilung im "klassischen Entwicklungszyklus" der kaum erfaßte Aufwand für das Erlernen und Üben von Programmiersprachen und "Softwaretools".

Phase	Aufwand ("klassische" im Vergleich zu KomponentenSW)
Systemanalyse	≤
Systemspezifikation	≥
Grobentwurf	>
Feinentwurf	>
Codierung	>>
Modultest	>>
Integration und Systemtest	≥
Entwicklungskosten	>>
Wartung und Pflege	??
Lebensdauerkosten	??

Tab.5: Erwartete Änderungen im Lebensdauerzyklus

3.3 Lösungsansätze

3.3.1 Technische Maßnahmen

Ein zunächst bestechend erscheinender Lösungsansatz wird in Diskussionen zum dargestellten Thema immer wieder genannt: die Verwendung von **"public domain software"** (z.B. Linux) anstelle kommerziell verfügbarer Produkte. Als hauptsächlicher Vorteil erscheint dabei meist die Verfügbarkeit des Quellcodes, von der man sich verspricht, die eingesetzten Programme genau so unter Kontrolle zu haben wie selbst entwickelte. Bei genauerer Betrachtung stellt sich dies als Illusion heraus. Die Erfahrung lehrt, daß langfristige Nutzbarkeit und Wiederverwendbarkeit von (auch selbst entwickelter) Software entscheidend von der Qualität der Dokumentation abhängt. Diese ist aber üblicherweise unzureichend. Es gibt keinen Grund, warum dies bei (kostenlos) öffentlich bereitgestellter Software anders sein sollte. Unabhängig davon bleibt eine andere Schwierigkeit bestehen: selbst wenn die verwendete Fremdsoftware optimal dokumentiert ist, muß sie im Falle von Änderungen oder Weiterentwicklungen jedes Mal neu analysiert, verstanden und auf ihre Auswirkungen auf das restliche System hin bewertet werden.

Erfolgversprechender erscheint es daher, beim Entwurf von Systemen aus Standardkomponenten von vornherein die **Systemarchitektur** besser durchzudenken. Insbesondere sollten die vorauszusehenden Schnittstellen genauestens analysiert und ihre Anzahl minimiert werden. Sehr hilfreich kann es auch sein, den Datenverkehr über die inneren Schnittstellen auf das notwendige Minimum zu beschränken ("lose Kopplung"). Bei der Entwicklung grosser Systeme im kaufmännischen und Verwaltungsbereich wird auch manchmal das Verfahren

eingesetzt, selbst definierte Zwischenschnittstellen einzuführen, die man folglich entwicklungstechnisch im Griff hat. Dieses Vorgehen ist aber in der Automatisierungstechnik mit Vorsicht zu betrachten - es kann viel Laufzeit kosten.

Das Wichtigste scheint aber die **komplettere und tiefergehende Analyse verfügbarer Komponenten** zu sein. Im Lebensdauerzykus sollte sie das ergänzende Gegenstück zu einer sachgerechten Systemanalyse darstellen. Ein wichtiges Hilfsmittel dabei kann die Bestimmung des Langlebigkeitsindex [Lö02] sein.

3.3.2 Managementmaßnahmen

Unter Managementgesichtspunkten wird häufig die Verwendung von **"public domain software"** als ein Ausweg aus der Abhängigkeit von übermächtigen Komponentenlieferanten betrachtet. Jedoch scheint die sich dann ergebende Abhängigkeit von einer "sich selbst regulierenden" Gruppierung keine Verbesserung darzustellen. Im Gegenteil: tritt durch Fehler in der verwendeten Software ein Haftungsfall ein, so steht der Systemlieferant völlig allein. Zweckmäßiger erscheint eine sorgfältige Vertragsgestaltung mit den Lieferanten.

Im Innenverhältnis muß das "klassische Projektmanagement" durch weitere Maßnahmen ergänzt werden. Dabei bieten sich u.A. **Schnittstellenmanagement** und **Konfigurationsmanagement** an. Diese beiden Ansätze müssen aber noch weiter entwickelt werden, um den besonderen Anforderungen zu entsprechen, die sich aus der dargestellten Problematik ergeben. Auch die Einführung selbst definierter Zwischenschnittstellen erscheint aus Managementgesichtspunkten sehr erfolgversprechend. Es ist aber zu bedenken, daß sie in der Regel den Entwicklungsaufwand erhöhen werden.

Aus der Bestimmung des **Langlebigkeitsindex** [Lö02] müssen managementorientierte Konsequenzen gezogen werden, wie z.B. eine geeignete Bewertung der Lieferfirmen. Als ein mögliches Vorbild kämen z.B. Überlegungen analog zum SEI-Prozeßmodell in Frage.

Nicht zuletzt ist eine **neuartige Schulung der Mitarbeiter** notwendig, die bei diesen Problembewußtsein bezüglich der neuen Herausforderungen schafft.

3.3.3 "Branchenpolitische" Maßnahmen

Schließlich ist der Verfasser der Meinung, daß die dargestellte Problematik durch altbewährte handels- oder "branchen"politische Maßnahmen schon zu einem erheblichen Teil entschärft werden kann. Auf die Gefahr hin, "Eulen nach Athen zu tragen", sollen einige davon hier kurz aufgelistet werden:

- Schaffung verbindlicher firmenübergreifender Standards für Übergabeformate,
- Offenlegung existierender Firmenstandards,
- Offenlegung geeigneter Bewertungen von Lieferfirmen.

3.4 Datenerhebung

Aus dem Gesagten dürfte klar geworden sein, daß der Verfasser nicht der Meinung ist, daß es für die Probleme, die sich in dieser neuen Phase der Entwicklung der Softwareindustrie bemerkbar machen, schon eine Lösung gibt. Es wird aber nicht nötig sein, völlig neue Ansätze zu suchen, wenn die bisherigen Vorgehensweisen und "Modelle" ergänzt und an die neue Situation angepasst werden. Man sollte aber einmal nicht nach dem Motto vorgehen: "Erst postulieren, dann ausprobieren“, sondern zunächst Erfahrungen, Fakten und Zahlen sammeln.

Für dieses Vorhaben scheinen die existierenden Gruppierungen zum Thema „Management der Softwareentwicklung“ hervorragend geeignet. Der Verfasser dankt deshalb für die gebotene Diskussionsmöglichkeit und Unterstützung und würde sich über möglichst viele ausgefüllte Fragebögen sehr freuen.

4 Literaturhinweise

[Br75] Brooks, F. P. Jr.: The Mythical Man Month; Addison Wesley, 1975

[El04] Elzer, P.: Software aus Standardkomponenten - wo stecken neue Probleme? WI-MAW-Rundbrief des GI-Fachausschusses "Management der Anwendungsentwicklung und -wartung" (im Druck)

[El97] Elzer, P.: SW-Management im Wandel. in: Oberweis/Sneed (Hrsg.): Software-Management '97, Teubner-Verlag, München, 29.-31. Oktober 1997, S. 48 - 62

[EL02] Elzer, P.; Löbbert, A.: Ein Verfahren zur Beurteilung der langfristigen Nutzbarkeit von Softwaresystemen; GI-Konferenz Software Management 2002 "Fortschritt durch Beständigkeit", GI-Edition Lecture Notes in Informatics, Hamburg, 5. - 8.11.2002, S. 99-110

[LE00] Löbbert, A., Elzer, P.: Lehren und Lernen über das Internet; GMW-Tagung "Campus 2000: Lernen in neuen Organisationsformen (Medien in der Wissenschaft, Band 10), Innsbruck, 19.- 21.09. 2000, S. 405-406

[LE99] Löbbert, A., Elzer, P.: Entwicklung eines Webgestützten Lehr- und Lernsystem; Vortrag gehalten bei Informatik '99 Paderborn, Oktober 1999, veröffentlicht unter http://www11.informatik.tu-muenchen.de/~brueggem/neueMedien99/

[Lö02] Löbbert, A: Beurteilung der Eignung von Softwaresystemen für eine lange Lebensdauer; Dissertation; Fakultät für Bergbau, Hütten- und Maschinenwesen; Papierflieger, Clausthal-Zellerfeld, ISBN 3-89720-608-0, 2002

Anhang: Fragebogen

Vorbemerkung: die folgenden Fragen mögen vielleicht manchmal etwas "unscharf" erscheinen. Sie wurden aber absichtlich so formuliert, um a) die Anonymität sicherzustellen und b) den Aufwand für die Beantwortung gering zu halten.

Blatt A: Fragen zum Gesamtsystem

A.I Branche, in der oder für die das System entwickelt wurde: . . . . . .

A.II Quantitatives zur Technik

II.1 Größe des Gesamtsystems (in LSC oder, falls nicht bekannt, in LOC) . . . . . .

II.2 Größe des selbst entwickelten Anteils (s.o.) . . . . . .

II.3 Anzahl der Fremdkomponenten . . . . . .

II.4 Anzahl der internen Schnittstellen im Gesamtsystem (zwischen allen Komponenten),
die bei Entwicklung (und Wartung) beachtet werden mußten (müssen): bei der Entwicklung: . . . . . .
während der Wartung: . . . . . .

II.5 Kenngrößen der Fremdkomponenten
(hierzu bitte für jede Fremdkomponente einen Bogen "B" ausfüllen)

A.III Quantitatives zum Aufwand

III.1 Gesamtentwicklungsaufwand (z.B. in Mannmonaten) . . . . . .

III.2 Entwicklungsaufwand für "Contents" (Datenbestände, Grafiken, etc.) . . . . . .

III.3 Beobachtete Kostenverteilung auf die Entwicklungsphasen
(überschlägig geschätzt in % des Gesamtaufwands)
Systemanalyse: . . . . . .
Systemspezifikation: . . . . . .
Grobentwurf: . . . . . .
Feinentwurf: . . . . . .
Codierung: . . . . . .
Modultest: . . . . . .
Integration und Systemtest: . . . . . .

III.4 Mittlerer Aufwand zur Nachführung des Gesamtsystems bei Änderung einer Komponente
durch deren Hersteller (z.B. in MM): . . . . . .

A.IV Dynamik des Gesamtsystems

IV.1 Beobachtete oder erwartete Lebensdauer des Systems: . . . . . .

IV.2 Beobachtete oder erwartete Lebensdauer der Contents (vergl. III.2): . . . . . .

IV.3 Anzahl Releases pro Projektdauer / Lebensdauer der Anwendung: . . . . . .

IV.4 Störungen der Entwicklung durch Änderung der Eigenschaften von Komponenten: . . . . . .

A.V Erfahrungen mit Lösungsansätzen

V.1 Eingesetzte technische Maßnahmen: . . . . . .

V.2 Ergriffene organisatorische Maßnahmen im Innenverhältnis . . . . . .

V.3 Ergriffene organisatorische Maßnahmen im Außenverhältnis: . . . . . .

Blatt B: Fragen zu Fremdanteilen (bitte für jede Komponente ein Blatt benutzen)

B.1 Fremdkomponente **Nr.: . . . . .**

B.2 Aufgabe: . . . . . .

B.3 Größe (in LSC oder, falls nicht bekannt, in LOC): . . . . .

B.4 Anzahl Funktionen (falls bekannt):

B.4.1 gesamt: . . . . .

B.4.2 für Anwendung benötigt: . . . . .

B.5 Lernaufwand (z.B. in Mannwochen): . . . . .

B.6 Aufwand zur Anpassung an Anwendung (falls nötig oder möglich): . . . . .

B.7 Schnittstellen zu Komponenten Nr.: . . . . . .

B.8 Zum Betrieb benötigte andere Fremdkomponenten): . . . . . .
(hierzu bitte wieder für jede Fremdkomponente einen Bogen "B" ausfüllen)

B.9 Bei Änderungen betroffene andere Komponenten: . . . . . .

B.10 Änderungsfrequenz (Mittlere Häufigkeit von Releases der Komponente): . . . . .

B.11 Nachführaufwand pro Änderung (z.B. in Mannwochen): . . . . .