*Design for Future**

Legacy-Probleme von morgen vermeidbar?

Prof. Dr. Gregor Engels
Institut für Informatik, Universität Paderborn,
Paderborn, Deutschland

Prof. Dr. Michael Goedicke
Institut für Informatik und Wirtschaftsinformatik,
Universität Duisburg-Essen, Essen, Deutschland

Prof. Dr. Ursula Goltz
Institut für Programmierung und Reaktive Systeme,
Technische Universität Braunschweig,
Mühlenpfordtstr. 23, 38106 Braunschweig, Deutschland
E-Mail: goltz@ips.cs.tu-bs.de

Prof. Dr. Andreas Rausch
Institut für Informatik, Technische Universität Clausthal,
Clausthal-Zellerfeld, Deutschland

Prof. Dr. Ralf Reussner
Fakultät für Informatik, Universität Karlsruhe (TH),
Karlsruhe, Deutschland

Die Herausforderung

Eine weit verbreitete Annahme ist, dass Software nicht altert, da Software als immaterielles Gut keinerlei Verschleißerscheinungen unterliegt. Diese Annahme ist aber falsch – Software altert. Die Umgebung, in der Software eingesetzt wird, verändert sich kontinuierlich und die dazugehörige Hardware und die Infrastruktur verändern sich. Wird die Software nicht ebenfalls kontinuierlich weiterentwickelt, so altert sie rasch relativ zu ihrer Umgebung. Dieses Problem ist vor allem im Bereich der großen betrieblichen Informationssysteme unter dem Oberbegriff Legacy bekannt. Es sind große Systeme im industriellen Einsatz, teilweise noch in FORTRAN oder COBOL geschrieben, die trotz suboptimaler Funktionalität und Qualität weiter unangepasst im Einsatz bleiben, da eine Optimierung an der Komplexität des Systems und dem Aufwand für eine Einarbeitung durch die heutigen Entwickler scheitert. Eine Neuerstellung solcher Systeme würde enorme Kosten und Probleme bei der Umstellung verursachen.

Dieses bekannte Problem wird sich noch weiter verschärfen: Zum einen gewinnen die eingebetteten Systeme immer größere Bedeutung. Hier sind wir mit der Situation konfrontiert, dass aufwändige Software in langlebigen technischen Geräten eingesetzt wird. Zu welchen Problemen dies führt, zeigt sich zum Beispiel bei Systemen für den Betrieb der Space-Shuttle-Raumtransporter. Wie die New York Times am 12. Mai 2002 berichtete [7], ist es für die Nasa wesentlich günstiger, den verfügbaren Markt nach (fast) obsoleter Hardware abzusuchen bzw. gerade noch verfügbare Komponenten aufzukaufen, als die damit verknüpfte Software neu zu entwickeln oder auf neue Plattformen zu migrieren.

Darüber hinaus macht die zunehmende Vernetzung von Systemen die Situation noch schwieriger. Systemgrenzen können oft nicht mehr klar definiert werden. Es ist manchmal nicht mehr möglich, vollständiges Wissen über das Gesamtsystem bereitzustellen. Trotzdem müssen ganze Anwendungslandschaften beherrscht werden. Das Ausschalten von Teilsystemen ist oft aufgrund der Vernetzung nicht möglich. Alterungsprozesse von Teilen gefährden das Gesamtsystem; dies lässt sich plakativ mit dem Begriff „Erosion von Softwaresystemen“ beschreiben.

Notwendige Updates in existierenden, rund um die Uhr laufenden Softwaresystemen führen dann immer wieder zu erheblichen Problemen. Aktuelles Beispiel ist ein Softwareausfall Ende April 2009 beim Mobilfunkanbieter T-Mobile, dessen Netz für mehrere Stunden für Sprach- und SMS-Dienste nicht nutzbar war. Wiederholt wurde hierüber in der Presse berichtet und mitgeteilt, dass die genaue Ursache des Softwarefehlers den Verantwortlichen nicht klar war. Für viele Kunden von T-Mobile war der Softwareausfall nicht nur mit Unannehmlichkeiten verbunden, sondern etwa im Geschäftsverkehr im Einzelfall auch mit erheblichem finanziellem Schaden.

Das Problem hat also enorme ökonomische Bedeutung für unsere Wirtschaft und Gesellschaft. Die Situation wird sich in Zukunft weiter verschärfen, da die Spanne zwischen der Geschwindigkeit des technologischen Fortschritts und der Anpassbarkeit von Anwendungen an wechselnde Plattformen immer größer wird. Der Umgang mit Legacy-Software wird immer wieder diskutiert, ohne dass die Probleme wirklich gut gelöst werden können. Es besteht dringender Handlungsbedarf. Aus unserer Sicht ist es unbedingt erforderlich, einen neuen Ansatz in der Softwaretechnik zu entwickeln, um die

* Erschienen im Informatik Spektrum, Springer-Verlag 2009, DOI 10.1007/s00287-009-0356-3. Die ursprüngliche Veröffentlichung ist erhältlich über www.springerlink.com

Entstehung immer neuer Legacy-Probleme zu vermeiden. Dazu analysieren wir zunächst, wie es zum Entstehen von Legacy-Systemen kommt und zeigen dann mögliche Lösungsansätze auf.

Aus unserer Sicht wird bei der Erstellung von Softwaresystemen, auch bei großen und langfristig angelegten Projekten, zu kurzfristig gedacht. Es werden hohe Qualitätsansprüche gestellt und mit geeigneten Methoden realisiert, aber immer mit dem Blick auf den unmittelbaren und mittelfristigen Einsatz des erstellten Systems. Änderbarkeit und Wartbarkeit werden durchaus berücksichtigt und angestrebt. Aber kein Softwareingenieur denkt heute bei der Systementwicklung wirklich daran, dass das System, das er erstellt, vielleicht in 20 oder 30 Jahren immer noch laufen wird, und wenn, könnte er sich nur schwer vorstellen, wie die Randbedingungen dann aussähen. Prominentes Beispiel ist das Jahr-2000-Problem: Die Programmierer in den 1970er- und 1980er-Jahren haben nicht in Erwägung gezogen, dass ihre Systeme im Jahr 2000 noch im Einsatz sein könnten. An dieser Situation hat sich trotz dieser Erfahrung bisher nichts grundlegend geändert. Auf der Metaebene, bei der Entwicklung von Methoden und Werkzeugen in der Softwaretechnik, sehen wir das gleiche Problem. Auch hier werden derzeit noch kaum Methoden für die Unterstützung des wirklich langfristigen Einsatzes von Software entwickelt. Die Evolution von Systemen, die Anpassung an sich verändernde Anforderungen und Randbedingungen, wird bisher wenig unterstützt. Insbesondere die Anpassbarkeit an wechselnde Plattformen erfordert ganz neue Ansätze. Neue Trends in der Softwaretechnik können unter Umständen die bestehenden Probleme sogar verschärfen. Serviceorientierung kann zur Zementierung von Schnittstellen führen und basiert gleichzeitig auf einem komplexen Technologie-Stack. Letzteres führt offensichtlich zu Problemen, wenn die Lebenszyklen für die Schichten divergieren. Die beabsichtigte schnelle (Weiter-)Entwicklung von Systemen mithilfe von im Internet verfügbaren Services stellt die Softwareentwicklung somit vor weitere Herausforderungen. Einerseits wird hohe Flexibilität durch „freie“ Auswahl von gleichwertigen Implementierungen eines Service gewonnen, und andererseits muss sichergestellt werden, dass das in der realisierten Anwendung vorhandene Wissen über Geschäftsprozesse bei der gewünschten Weiterentwicklung ebenfalls eine entsprechende Evolution erfährt.

Um die Entstehung immer neuer Legacy-Systeme zu vermeiden, muss ein grundlegendes Verständnis für die Problematik der Entwicklung langlebiger zukunftsfähiger Softwaresysteme entwickelt werden. Im Folgenden möchten wir aufzeigen, was wir auf der Forschungsseite beitragen können, um die erheblichen Investitionen, die in große Softwaresysteme getätigt werden, zu schützen und massive Probleme durch zunehmende Softwareerosion zu verhindern.

Aktuelle Ansätze in der Softwaretechnik, insbesondere im Bereich Softwarearchitektur, können dazu beitragen, die Situation zu verbessern, wenn sie geeignet weiterentwickelt und angewandt werden. Es ist eine spannende und wichtige Herausforderung, diese Ansätze zu integrieren und auf die Langlebigkeit und die Evolution von Software auszurichten. Die Vision ist „Forever-Young-Software“ zu bauen, die an sich verändernde Anforderungen und Randbedingungen mit vertretbarem Aufwand anpassbar ist oder sich sogar selbst anpasst.

Lösungsansätze

Zunächst stellen wir fest, dass die herkömmlichen Vorgehensmodelle mit der klassischen Trennung zwischen Entwicklung einerseits und Wartung bzw. Modifikation bestehender Systeme andererseits für langlebige Systeme nicht mehr adäquat sind. Softwareentwicklung muss sich konzeptionell über den gesamten Lebenszyklus hinweg fortsetzen. Koevolution zwischen Anforderungen, Architektur und Implementierung sowie auf höherer Ebene zwischen Entwicklung, Deployment und Betrieb der Software ist erforderlich. Entsprechend muss Koevolution zwischen Anwendungen und Entwicklungsumgebungen und Werkzeugen stattfinden.

Das wichtigste Paradigma für die Entwicklung langlebiger zukunftsfähiger Softwaresysteme ist die Modellbildung in allen Bereichen, insbesondere für Requirementsengineering und Softwarearchitektur. Gerade der zentrale Stellenwert der Softwarearchitektur muss nicht nur, aber insbesondere auch im Bereich eingebetteter Systeme, noch mehr in die Praxis getragen werden. Die Softwaretechnik stellt mit dem architekturbasierten und modellgetriebenen Vorgehen eine gute Basis bereit. Was aus unserer Sicht bisher jedoch fehlt, sind Ansätze, um Architektur und Modelle mit ausreichend Information zu versehen, um die Weiterentwicklung komplexer Softwaresysteme zu ermöglichen. Analog zum Ansatz des Proof Carrying Code (PCC [10]) müssen Softwaresysteme auch auf der Modell- und Architekturebene das eigene Metawissen enthalten bzw. es müssen Mechanismen gefunden werden, die eine enge Verbindung zwischen Modell und Realisierung ermöglichen. Ein vergleichbarer Ansatz scheint sich unter dem Begriff Specification Carrying Code zu entwickeln [4] oder durch in Programme eingebettete Modelle (siehe z. B. [1, 3, 12]). Auf der Ebene der Softwarearchitektur müssen geeignete Erweiterungen der Modellierungsansätze entwickelt werden. Diese bilden das Handwerkszeug, um die im Folgenden diskutierten Aspekte zu unterstützen.

Für die Evolution von Systemen ist es sehr wichtig, das System immer in seinem Kontext zu sehen. Dazu gehört zum Beispiel die Begründung von Architekturentscheidungen. Für die Qualitätssicherung über den Lebenszyklus hinweg müssen die notwendigen Informationen bereitstehen, unter anderem durch Integration von Test- und Review-Dokumenten. Nachverfolgbarkeit durch Verlinkung, zum Beispiel zwischen Anforderungen und Architekturentscheidungen, muss gewährleistet werden (siehe z. B. [6, 9]). Besonders wichtig ist die Handhabung von Variabilität. Die Arbeiten aus dem Bereich der Produktlinienansätze [11] haben gezeigt, dass die vorausschauende Handhabung von Variabilitäten einen erfolgversprechenden Ansatz für die Beherrschung bestimmter Komplexitäten im Requirements-Management darstellt. Die vorhandenen Ansätze müssen geeignet weiterentwickelt werden, um die Darstellung und das Management von Vielfalt und Variabilität auf der Anforderungsebene und der (Software-)Architekturebene gemeinsam voran zutreiben. Entsprechende Prozesse für das Anforderungsmanagement sind zu definieren, die Architektur, die Realisierung und der Einsatz sind übergreifend über den gesamten Lebenszyklus hinweg zu beherrschen.

Im Bereich der Architektur bilden Muster für die Problem- und Systemebene [8] einen guten Ausgangspunkt. Die in diesen Problembereichen bisher entstandenen Ansätze für die eher kleinteiligen Ansätze zur Beschreibungs- und Entwicklungsmethodik sind für den Umgang mit Systemen von Systemen und auf der Ebene von Softwarelandschaften (EAMs) durch adäquate Konzepte und Methoden zu erweitern. Entsprechende Werkzeuge wären dann in der Lage, die Probleme der stetigen Veränderung und des Betriebs solcher Anwendungslandschaften besser zu beherrschen.

Die Repräsentation der den Softwaresystemen innewohnenden Konzepte, Funktionen, Daten und sonstigen Eigenschaften in Form geeigneter Modelle (Modellbasierung) spielt eine zentrale Rolle, da die ablauffähige Version eines Softwaresystems zu viele für die hier betrachtete Ebene unwichtige Details enthält. Mit einer adäquaten Darstellung der Modelle von Softwaresystemen wird es möglich, die Entscheidungen in der Weiterentwicklung durch strikte und formale What/If-Analysen zu unterstützen. Für bestehende Systeme wird es nötig sein, Modellextraktion durch Program Comprehension zu realisieren. Dazu werden zurzeit modellbasiertes Reengineering und Refaktoring als aktuelle Forschungsthemen behandelt. Ergebnisse in diesen Bereichen werden die integrierte Entwicklung auf Modell- und Codeebene sowie die Integration von neu entwickelter Software in Systemlandschaften unterstützen.

Die Frage stellt sich natürlich, wie in einer solchen komplexen Entwicklungssituation die Qualitätssicherung sicher gestellt werden kann. Ähnlich wie Requirements Engineering über den gesamten Lebenszyklus sichergestellt werden muss, ist die Realisierung verschiedener Qualitätsziele über den gesamten Lebenszyklus hinweg nur durch entsprechende Monitoring-Konzepte zu erzielen. Auf der Basis von Modellen können auch hier geeignete architekturbezogene Informationsbedarfe und Bewertungsmaßstäbe definiert werden und über entsprechende Techniken bereitgestellt werden. Neben „klassischen“ Qualitätszielen wie Leistung und Sicherheit ergeben sich besondere Herausforderungen für strukturbezogene Maßstäbe wie Wartbarkeit und Weiterentwicklungsfähigkeit. Die empirische Absicherung solcher Maßstäbe ist unabdingbar, um zu verlässlichen Aussagen zu kommen, die zielgerichtet zu Entwicklungsentscheidungen führen können.

Aufgrund der Größe der Probleme und Systeme, die betrachtet werden, ist es selbstverständlich notwendig, dass alle hier skizzierten Ansätze und Methoden durch Werkzeuge unterstützt werden müssen. Zusätzlich ergibt sich das Problem der Skalierbarkeit der Ansätze, da sich leicht für die stark vernetzten Systeme (zu) lange Analysezeiten ergeben, falls eine bestimmte Größenordnung überschritten wird. Als Beispiel für erfolgreiche Werkzeuge für die eher detaillierte Analyse der Struktur eines Softwaresystems sei hier die Software-Tomografie [2] genannt. Eine Fortsetzung vergleichbarer Ansätze für Systeme von Systemen ist notwendig.

Des Weiteren bietet die schnelle Entwicklung im Bereich der IT-Technologien zurzeit zusätzliche Fallstricke. Die Festlegung auf ein bestimmtes Framework oder eine bestimmte Plattform kann schnell das Aus für ein Softwaresystem bedeuten, wenn beispielsweise der Technologielieferant entscheidet, sein Produkt nur in veränderter Form oder gar nicht mehr zu liefern. Diese Plattformfalle kann im Prinzip durch virtuelle Maschinen – ähnlich wie im klassischen Compilerbau – entschärft werden, wenn eine geeignete Form der Interaktion zwischen verschiedenen Applikations-/Systemteilen gefunden werden kann.

Alle Methoden und Techniken können am Ende nur dann überzeugen, wenn es gelingt, sie so aufzubereiten und zur Verfügung zu stellen, dass die Entwicklungsteams in die Lage versetzt werden, die Funktionen und Methoden solch mächtiger Entwicklungsumgebungen auch anzuwenden. Aspekte der Organisation und Kommunikation dürfen nicht unterschätzt werden. Die Verteilung komplexer Entwicklungsprobleme auf die Schultern verschiedener Entwicklungsteams ist immer noch eine große Herausforderung, insbesondere, wenn es darum geht, gemeinsam in einem verteilten Team komplexe Problemstellungen zu modellieren und zu analysieren. Die Herausforderungen liegen hier auch in der Darstellung und Interaktion für verteilte Teams. Auch hier sind empirische Untersuchungen notwendig, um die Effektivität und Effizienz solche Ansätze zu bewerten.

Weiterhin ist es notwendig, dass eine enge Verzahnung zwischen Business und IT erfolgt, um sicherzustellen, dass Softwaresysteme (weiter-)entwickelt werden, die die tatsächlichen Bedürfnisse des Endanwenders erfüllen. Hierzu gehören neuartige Organisationsformen wie die enge Zusammenarbeit von Domänenexperten und Softwareentwicklern etwa in agilen Entwicklungsprojekten oder auch neuartige Softwareentwicklungsmethoden, die bei der Analyse von Anforderungen an die Software bereits bei strategischen Geschäftszielen und Geschäftsprozessen ansetzen. Dies kann dann zu neuartigen Softwarearchitekturstilen (etwa serviceorientierten Architekturen (SOA)) führen, deren Komponentenstruktur unmittelbar Geschäftsdimensionen und Geschäftsfelder widerspiegelt und dadurch vorbereitet ist auf sich verändernde Anforderungen des zu unterstützenden Geschäfts [5].

Was müssen wir tun?

Um das Entstehen neuer Legacy-Systeme zu vermeiden, muss einerseits ein Umdenken bei Softwareentwicklern, bei Projektleitern und im Management stattfinden. Es muss erreicht werden, dass die Lebenszeit von Software realistisch eingeschätzt wird und die entsprechend notwendigen Maßnahmen bei der Softwareerstellung als essenzieller Bestandteil gesehen werden. Andererseits müssen durch intensive Forschungstätigkeiten im Bereich der Informatik und insbesondere der Softwaretechnik die Voraussetzungen für diese Maßnahmen geschaffen werden. Wenn sich die Bereiche, die hier Lösungsansätze beitragen können, mit dem Fokus auf langlebige Softwaresysteme koordinieren, können gezielt in gemeinsamer Arbeit die geeigneten Methoden, Werkzeuge, und Standards entwickelt werden.

Wir schlagen vor, als erfolgversprechenden Ansatzpunkt Metamodelle für die Integration von Wissen über Software im Bereich Softwarearchitektur, aber ebenso in anderen Bereichen wie etwa für das Testmanagement, zu schaffen. Darauf aufbauend können durch entsprechende Grundlagenforschung Methoden, Metriken, Werkzeuge und Prozesse entwickelt werden, die die Koevolution von Systemen mit ihren sich verändernden Anforderungen, Randbedingungen und Plattformen ermöglichen. Empirische Untersuchungen sichern die Validierung der Ergebnisse.

Was können wir erreichen?

Wir haben hier aufgezeigt, dass Alterung und Erosion von Softwaresystemen gefährlich und ökonomisch sehr problematisch sind. Wir haben aber auch gezeigt, dass es vielversprechende Lösungsansätze und eine sehr gute Perspektive durch entsprechende Fokussierung gibt.

Durch den hier skizzierten Lösungsansatz erreichen wir unter anderem

– Investitionsschutz für aufwändige Softwareprojekte,
– bessere Softwarequalität über den gesamten Lebenszyklus,
– Transparenz von Softwareentwicklungsprozessen,
– Nachvollziehbarkeit von Entscheidungen und Aktivitäten im Softwarelebenszyklus und
– geringeren Verlust von Wissen und damit bessere Wiederverwendung.

Wir sind optimistisch, dass mit gemeinsamer Anstrengung das Ziel evolutionärer langlebiger Softwaresysteme mit guter Qualitätssicherung über den gesamten Lebenszyklus erreichbar ist.

Literatur

1. Balz M, Striewe M, Goedicke M (2009) Embedding Behavioral Models into Object-Oriented Source Code. In: Liggesmeyer P (ed) Proceedings of Software Engineering 2009. Gesellschaft für Informatik e. V. (GI), Bonn, pp 51–62
2. Beyer D, Noack A, Lewerentz C (2005) Efficient relational calculation for software analysis. IEEE Trans Softw Engin 31(2):137–149
3. Büchner T, Matthes F (2006) Introspective Model-Driven Development. In: Gruhn V, Oquendo F (eds) Software Architecture, Third European Workshop, EWSA 2006, vol 4344. Springer, Berlin Heidelberg, pp 33–49
4. Di Marzo Serugendo G, Fitzgerald J, Romanovsky A,Guelfi N (2007) A metadatabased architectural model for dynamically resilient systems. In: SAC '07: Proceedings of the 2007 ACM symposium on Applied computing, ACM, New York, pp 566–572
5. Engels G, Hess A, Humm B, Juwig O, Lohmann M, Richter J-P, Voß M, Willkomm J (2008) Quasar Enterprise: Anwendungslandschaften serviceorientiert gestalten. Dpunkt, Heidelberg
6. Florentz B, Goltz U, Braam J, Ernst R, Saul T (2007) Architekturevaluation: Qualitätssicherung in frühen Entwicklungsphasen. In: Achtes Symposium AAET 2007 – Automatisierungs-, Assistenzsysteme und eingebettete Systeme für Transportmittel, Braunschweig, S 238–248
7. For Parts (2009) NASA Boldly Goes on eBay. http://www.nytimes.com/2002/05/12/us/for-parts-nasa-boldly-goes-on-ebay.html (Zugriff am 30. April 2009)
8. Jackson M (2001) Problem Frames: Analysing and Structuring Software Development Problems. Addison-Wesley, Boston, MA
9. Lochau M, Müller TC, Detering S, Goltz U, Form T (2009) Architektur-Evaluation von AUTOSAR-Systemen: Adaption und Integration. 1. Elektronik Automotive Congress, 19. Mai 2009, München
10. Necula GC (2000) Proof-carrying code (abstract): design, implementation and applications. In: PPDP '00: Proceedings of the 2nd ACM SIGPLAN international conference on Principles and practice of declarative programming, ACM 2000, New York, pp 175–177
11. Pohl K, Böckle G, van der Linden F (2005) Software Productline Engineering, Foundations, Principles and Techniques. Springer, Berlin Heidelberg
12. Rausch A, Broy M (2002) Evolutionary Development of Software Architectures. In: Technology for Evolutionary Software Development, a Symposium organised by NATO's Research & Technology Organization (RTO), Bonn