Komplexität als Ursache von Fehlern in und Risiken durch Software

Britta Schinzel[1]

Die moderne Gesellschaft bedient sich der Informationstechnik zur Bewältigung komplizierter Probleme und zur Steuerung komplexer Abläufe, d.h. zur Reduktion von Komplexität. Auf der anderen Seite treten jedoch durch die Software-technische Vermittlung und die durch sie ermöglichte Verdichtung von Prozessen neue Komplexitätsphänomene und -probleme auf den Plan, ja die Dynamik der Veränderung wird selbst zum Anpassungsproblem, zum Problem der Bewältigung von Komplexität, zum Risiko. Die IT-Ploblemlösungen beschleunigen nicht nur zeitlich, sie verdichten auch raumzeitlich die Kopplung zwischen Abläufen. Durch beide Effekte verursachen sie ihre eigenen Risiken und Zuverlässigkeitsprobleme. Dies gilt auch für Softwaresysteme selbst: die Dynamik der Hardware-Entwicklung hat immer komplexere Software-Systeme, interaktive Systeme und Organisation ermöglicht.Dadurch konnten immer mehr Anwendungsbereiche erschlossen werden und die Anwendungen integriert und vertieft werden.

Diese Entwicklung überholte bisher alle qualitätssichernden Innovationen, die - einerseits durch die Professionalisierung der Informatik und andererseits durch die verbesserten technischen Möglichkeiten selbst verfügbar - zur Bearbeitung der aus dieser Dynamik resultierenden Probleme entstanden. So wurden die Software-unterstützten Abläufe und die bestehenden Systeme nicht grundsätzlich zuverlässiger,vielmehr wächst die Zuverlässigkeit langsamer als die Innovationen (Schinzel 1996). Je größer die Systeme, je mehr Kontrolle an sie abgegeben wird, je umfassender die Anwendungsbereiche der Automatisierung, desto unzuverlässiger und unberechenbarer wird ihr Verhalten. Denn nicht nur wird die Komplexität der Software selbst dadurch erhöht, sondern auch die Komplexität der Anforderungen an die Umgebung (obgleich gerade die Reduktion der menschlichen Fehleranfälligkeit oft das Motiv für die Größe und Umfassendheit ist - eine paradoxe Wirkung). So hat sich z.B. gezeigt, daß Risiken entstehen können durch automatische Kontrollsysteme, die Umgebungsdaten falsch interpretieren, durch unvollständige Spezifikation der Software-Aufgabe oder durch Problemlösungen, die für die Anwender undurchschaubar bleiben.
Die Integration bestehender Systeme zu immer größeren Anwendungen, Datenverbünden und weltweiten Netzen läßt einerseits eine Verschärfung der wachsenden Komplexitäts- und Zuverlässigkeitsprobleme erwarten. Umgekehrt entlasten die durch die Vernetzung und Dezentralisation ermöglichten Entkoppelungen durch Isolierung und Ausschaltung von Fehlerbereichen auch von Komplexität und beschränken die Wirkungsbereiche von Unzuverlässigkeiten.

Komplexitätsprobleme also schlagen sich in Fehlerhaftigkeit von Software und in der

[1] Die hier aufgeführten Gedanken entstammen Diskussionen der TeilnehmerInnen des von mir geleiteten Diskus-Projekts “Komplexität, Erfahrung, Zuverlässigkeit” des FB 8 (Informatik und Gesellschaft) der GI.

Unkontrollierbarkeit der Wirkungen von Software in den durch sie unterstützten Zusammenhängen nieder. Damit natürlich auch in Unproduktivität und volkswirtschaftlichen Schäden. An dieser Situation hat sich, wie TA-Studien (etwa Schinzel 1996a) stets erneut zeigen seit 30 Jahren wenig geändert. W. Wayt Gibbs schreibt in seinem Artikel "Software´s Chronical Crisis"(im Scientific American, September 1994) über die anhaltende Softwarekrise: Aktuelle Studien zeigen, daß auf je 6 große Softwaresysteme, die zum Laufen gebracht werden, zwei kommen, die gestrichen werden müssen. Die durchschnittlichen Entwicklungskosten übersteigen die Pläne um die Hälfte, bei großen Projekten ist die Relation noch schlechter.
Es gibt also bis heute keine fehlerfreie, verläßliche Software. Wartung und Pflege sind deshalb unverzichtbar und extrem aufwendig. Und dennoch bieten sie keinen ausreichenden Schutz gegen Abstürze des Systems, riskante Abläufe und katastrophale Folgen.

Daher zeigt sich innerhalb der Informatik zunehmend die Orientierung auf Sicherheits- und Qualitätsfragen von Software. Die seit langem in die Entwicklung von Verifikationstechniken investierten Anstrengungen haben leider noch nicht zu befriedigenden Ergebnissen für umfängliche Softwarepakete geführt. Überdies treffen diese Bemühungen nicht den Großteil der vorfindlichen Fehler. Das sind nämlich jene, die durch unvollständige und falsche Spezifikation hervorgerufen werden. Daher werden auf der einen Seite kombinierte Qualitätssicherungskataloge mit Test- und Softwaremaßen angefertigt und ihre Erfüllung gefordert. Andererseits müssen für sicherheitskritische Software spezifische Methoden der Lokalisierung, Isolierung und Vereinfachung der sicherheitskritischen Anteile angestrengt werden, um diese Teile dennoch einer exakten sauberen Verifikation zugänglich zu machen. Dieses Vorgehen läßt sich aber nur dann durchführen, wenn in der Tat die sicherheitskritischen Schnittstellen feststellbar und isolierbar sind, wovon jedoch nicht generell ausgegangen werden kann (oft werden sie erst ex post klar, nachdem der kritische Fall eingetreten ist).

Die hier verfolgte Sicht auf die Zuverlässigkeitsproblematik von Software verlegt den Standort der Untersuchung allgemein auf das Komplexe, also auch auf Komplexitätsphänomene außerhalb des Formalen und Algorithmischen. Neue adäquatere Operationalisierungen und Maße sind erforderlich, um daraus Ansätze zur Beherrschung von Fehlern und Risiken zu erhalten.

Eine neue Komplexitätstheorie ist notwendig:

Es ist wahrscheinlich die wichtigste Errungenschaft der Theoretischen Informatik, Ressourcenprobleme axiomatisiert und klassifiziert zu haben. Komplexitätsmaße für Laufzeit- und Speicherplatzbedarf (Blums Axiome, Turingmaschinenkomplexität, strukturelle Komplexitätsklassen) ebenso wie solche für statische Programmeigenschaften (Blums Axiome) und Beschreibungskomplexität (Kolmogoroff-Komplexität) und schließlich solche für automatische Lernaufgaben (Vapnik-Chervonenkis-Dimension) sind durchaus differenziert genug, um vielfältige Komplexitätsphänomene in Informatik und Lerntheorie zu erfassen. Doch bei der Erstellung und Anwendung von Software greifen sie nur zum Teil [2].

[2] Dies ist darauf zurückzuführen, daß sie eher Berechnungs- oder Lernaufwand denn Komplexität messen.

Die Aufgabenermittlung für die Spezifikation, wie auch die Softwareerstellung und die spätere Einbindung von Computersystemen in Arbeitszusammenhänge erfordern die formale Erfassung von Ausschnitten hochkomplexer sozialer Realität. Insbesondere ist hierbei zu beachten, daß soziale Komplexität auf mehreren Ebenen berücksichtigt werden muß. Zum einen muß der Realitätsausschnitt, in dessen Rahmen die Problemlösung zu bewältigen ist, expliziert werden, um diesen formalisieren zu können, zum anderen müssen die Organisations- und Kommunikationsstrukturen, in die das Softwareprodukt eingebettet werden soll, untersucht werden, um eine sozial verträgliche und effiziente Einbettung zu gewährleisten.Betrachtet man auf der anderen Seite die Computersysteme, so stellt man fest, daß diese selbst immer komplexer und unüberschaubarer werden und was als 'Vereinfachung' sozialer Realität gedacht war, wird selbst komplex, unüberschaubar und unkontrollierbar. Weitere Komplexitätsphänomene entstehen durch die Interaktionen von sozialen und technischen Systemen. Einige Beispiele mögen diese unterschiedlichen Dimensionen von Komplexität verdeutlichen.

a). Die Erstellung von Software erfordert im allgemeinen sowohl die holistische als auch die detailgenaue Kenntnis riesiger Zusammenhänge und komplexer Anwendungsgebiete, also etwas, das selbst von Fachleuten des jeweiligen Gebietes nicht immer geleistet werden kann. Darüber hinaus sind Formalisierung und Automatisierung als solche methodisch universell, das heißt, in jedem Fall muß formalisiert und damit expliziert werden, um automatisieren zu können. Diese uniforme Methode kann den unterschiedlichen Lebensbereichen, in die Computersysteme eingebaut werden, nicht gerecht werden.

b). Da die Software in den Arbeitsprozeß eingegliedert wird, entsteht eine Wechselwirkung zwischen den abzubildenden sozialen Vorgängen und dem Computersystem - der Einsatz der Software erzeugt neue Organisationsstrukturen, die wiederum durch die Software abgebildet werden sollen. Diese Form der Selbstbezüglichkeit müßte schon bei der Erstellung der Software berücksichtigt werden. Doch ist dies aufgrund der Verwendung von Standardsoftware und der Schwierigkeit, solche selbstbezüglichen Veränderungen vorauszusehen, im allgemeinen nicht möglich.

c). Die Einsatzumgebung bzw. die Organisationsstruktur, in die das System eingebettet ist, legt die Bedingungen für die zu schaffenden Anschlüsse fest und kann somit die Nutzbarkeit von Computersystemen stark beeinträchtigen. Auch Altlasten machen die Herstellung von Anschlüssen für die neue Software mühsam und führen zu unzuverlässigem Verhalten des Gesamtsystems.

Klassifikation von zuverlässigkeitsrelevanten Komplexitätsphänomenen:

I. Aus der Perspektive des Software Engineering: die durch Komplexität der Systeme, die durch Software teil-modelliert werden sollen, und des Software-Entwicklungsprozesses hervorgerufenen Fehler und Risiken. System-Entwicklung, -Entwurf, und -Gestaltung, also Produktion von SW. Hierzu gehört etwa die Komplexitätserhöhung der Aufgabe durch Vorverlegung der Verantwortungnahme in den Entwicklungsprozeß.

II. Aus der Perspektive der Theoretischn Informatik und der SW- Qualitätssicherung: Komplexität des Produkts Software selbst (statisch: Größe, Modularisierung, Vernetztheit,

Schnittstellen, Softwaremetriken; dynamisch: Komplexitätsmaße, Nichtrobustheit des Diskreten)

III. Aus der Perspektive der Technikfogenabschätzung und der Techniksoziologie: Komplexität der Verwendung und der Wirkungen von Software in ihrer Anwendungs-Umgebung.

Einige Beispiele sollen die Einteilung deutlich machen:

ad I.
1. Fehler haben ihre Ursache zum größten Teil in Spezifikationsmängeln [3]: ungenügende Kenntnis des Requirement Engineers über den Anwendungskontext, Veränderungen dieses Kontexts machen die Spezifikation fehlerhaft und unvollständig. Dies gilt ebenso für die Schnittstelle zwischen Conputerlösung und dem sozialen oder technischen Einbettungskontext. Solche Fehler sind nicht nur die häufigsten, sondern auch die teuersten: sie müssen bis zum Ursprung durch jede Phase der Softwareentwicklung zurückverfolgt werden.
Demgegenüber werden Fehler oder Unzulänglichkeiten in Formalisierung, Algorithmen oder im Programm meist während der Entwicklung festgestellt und korrigiert oder gar nie (z.B. wenn ein besserer Algorithmus zu schnelleren Ausführungszeiten geführt hätte), Fehler in der Spezifikation, auch oft der Modellbildung, erst im nachhinein.
Solche Fehler in der Anforderungsermittlung werden zwar über das Pflichtenheft oft an die Verantwortung der Auftraggeber zurückgespielt, wobei sich hier jedoch ein neuer Trend abzuzeichnen beginnt, der die Pflichten der Entwickler weiter faßt.
Das prinzipielle Verfehlen der Software von Wirklichkeit durch die Nichterfaßbarkeit der Komplexität der Realität, die Unmöglichkeit der rationalen Rekonstruktion, durch die hermeneutischen Probleme und die dadurch bedingte Blindheit der Entwickler, durch die Lücke zwischen Nichtformalem und Formalem und die notwendige Vergröberung und Fixierung fließender Semantiken durch Formalisierung, bewirkt eine notwendige Inadäquatheit der Spezifikation. Diese drückt sich vor allem in Unvollständigkeit aus, die wiederum eine unzulässige Komplexitätsreduktion durch die Software-vermittelten Abläufe, etwa im Arbeitsbereich, bewirkt.

ad I.
2. Ein weiterer Ursachenkomplex für Fehler sind Modularisierung und Schnittstellen. Jeder Teil des Systems kann mit anderen Teilen in Interaktion treten, deshalb können triviale Ereignisse in nichttrivialen Systemen unvorhersehbare Ereignisketten auslösen. Für den einzelnen Fehler haben die Entwickler meist Vorsorge getroffen, auf ihn sind in der Regel auch die Bediener gefaßt. Daß jedoch mehrere Fehler gleichzeitig auftreten, hat niemand vorhergesehen und geplant. Die komplexen Interaktionen sind den Bedienern im kritischen Zeitraum meist undurchschaubar. Dies hat seinen Grund auch darin, daß in großen

[3] Mit der Spezifikation wird festgelegt, was das künftige System können soll. Sie entscheidet über den Wirklichkeitsausschnitt, der in einem Programm abgebildet werden soll. Ist dieses Wirklichkeitsmodell unvollständig, erfüllt das Programm zwar die spezifizierte Aufgabe, kann aber in den unvorhergesehenen Fällen zu Katastrophen in seiner Anwendungsumgebung führen.

Softwareprojekten gleichzeitig mehrere hundert Entwickler an einem Produkt arbeiten. Dazu muß das Projekt in einzelne Programmteile aufgegliedert und eine schier unüberschaubare Zahl von Schnittstellen definiert und angepaßt werden. Diese Schnittstellen sind sehr fehlerträchtig, weil viele Details für selbstverständlich gehalten und daher überlesen oder übersehen und gar nicht erst dokumentiert werden. Schnittstellendefinitionen sind praktisch nie vollständig. Hinzu kommt, daß während der Projektdurchführung eine Fülle von Änderungen erforderlich wird, die immer wieder zu neuen Anpassungen der Arbeitsorganisation zwingen. Der Informationsfluß und die Koordination zwischen den Beteiligten können oft nicht den vielfältigen Wechselbeziehungen der einzelnen Programmteile gerecht werden und sind daher eine ständige Fehlerquelle.

Die laufende Fehlerkorrektur konvergiert keineswegs zu fehlerloser Software, im Gegenteil: es ist eine alte Regel, selten auftretende und nicht allzu katastrophale Fehler unberührt zu lassen, da ihre Beseitigung die Gesamtsituation noch unkontrollierbarer macht. Der Entwurf wird uneinheitlich und komplizierter, die Dokumentation muß geändert werden und die vorher durch Erfahrung und Testen kontrollierbaren Verbindungen zu Umgebungsprogrammen und Hardware sind noch undurchsichtiger und riskanter geworden. Dadurch wird meist mehr Schaden (noch mehr und schlimmere Fehler) als Nutzen anrichtet.

Die Diskretheit des Mediums Software bedingt, daß nicht korrekte Programe chaotisch werden, d.h. kleine Abweichungen der Eingabedaten können riesige unkontrollierte Sprünge und Abstürze auf der Ausgabeseite bewirken. Unvollständigkeiten der Spezifikation bedingen ebenfalls unvorhersehbare Wirkungen von Software. Ein ähnlich chaotisches Verhalten verursacht die Nichtlinearität der Beziehungen zwischen einzelnen Moduln (d.h. die Unkontrollierbarkeit der Seiteneffekte auf Schnittstellenvariablen), und zwischen Programm, Umgebung und Hardware-Teilen. Dies sind prinzipielle Eigenschaften des Mediums, keine durch noch so gute Verifikationsmethoden und Testläufe - so es sie gäbe - kontrollierbare und abzusichernde Risikopotentiale.

ad I.
3. Der Software-Einsatz ändert Produkte, Prozesse, menschliche Arbeit, Menschen und die Beziehungen zwischen ihnen.
Software tritt ja nicht nur in Form für direkte Computernutzung in Erscheinung, sondern auch in Bauteilen von technischen Produkten, eingebettet in Produktionsprozesse und organisatorische Prozesse oder als solche Prozessabläufe steuernd. Dies macht Rationalisierungsprozesse möglich. Dadurch wird der der am Computer Arbeitende oft von routinemäßigen Arbeiten befreit. Das bewirkt die verstärkte Beschäftigung nur mit Ausnahmen, also mit ständig wechselnden Arbeitsvorgängen, die sich so wieder in einer Streßerhöhung niederschlagen.

Der Einsatz von Software verändert die Bedingungen des Einsatzes und damit die Korrektheit der Spezifikation, ein selbstbezüglicher Effekt, der die Möglichkeit zur Selbstorganisation notwendig macht, eine für Software nur eingeschränkt erfüllbare Bedingung.

Die Verstehensproblematik (Abstraktheit, Diskretheit, Komplexität) der informationstechnischen Abläufe und Funktionen erschwert die Verdeutlichung ihrer Wirkungen und Möglichkeiten und erhöht damit die Möglichkeit fehlerhafter Bedienung oder falscher Reaktion auf riskante Situationen.

ad II.
Aus der Techniksoziologie sind Ansätze bekannt (Perrow), die Komplexität und Kopplungsgrad der Anschlüsse zwischen einzelnen Teil-Abläufen als unabhängige Variablen mit dem Risikograd in Beziehung setzen. Strikte Kopplungen sind risikoreich in komplexen Zusammenhängen und lassen überdies eine Vielfalt von Anschlußmöglichkeiten nicht mehr zu.
In diesem Sinne sind z.B. die durch "Softwarezement" hergestellten festen Bindungen in komplexen Organisations- oder Produktionsabläufen risikoreicher als solche nicht festgelegte, wo die Selbstheilungsfähigkeit der Gesamtorganisation durch Selbstorganisation nicht behindert wird.
Die Problematik dieser 'Bindungskraft' äußert sich dann in der Spannung zwischen dynamischer, sozialer Realität und formalisierten Abläufen in Form von Software, die sich dadurch erklärt, daß das Formale, das heißt, Objektivierte, Formalisierte, Eindeutige feste Kopplungen in dem weichen Medium Realität, in das sie eingebettet ist, bildet, sie dadurch teilweise festzurrt und damit dynamische Veränderungen behindert.
Ein gutes Beispiel hierfür sind die Software-Altlasten, welche seit nunmehr dreißig Jahren durch ihre schiere Existenz die Folgesoftware an sich binden und die Freiheit der Wahl von besseren, verläßlicheren Programmiersprachen, Systemen und Problemlösungen einschränken.
Wegen der nicht mehr nachzuvollziehenden Unbekanntheit der ursprünglichen Entwurfs- und Programmierideen wird hier das Problem der Fehlerbehandlung zu einem rein empirischen, wenn nicht chaotischen.

ad III. G. Wohlands und D. Siefkes´Vorschläge, kleine überschaubare Systeme zu entwickeln, die zusammengesteckt, aber auch wieder entfernt werden können, erniedrigen den Kopplungsgrad zwischen Moduln, aber auch zwischen System und nichttechnischer Umwelt. In diesem Sinne wäre das ingeniuersmäßige Vorgehen mit der Komposition von Teilen aus einer ausreichend großen Basis-Menge normierter Bausteine zur gewünschten Programmleistung günstig, sofern sich Funktionsteile daraus herauslösen und ersetzen lassen und der Produktions- oder Organisationsablauf auch ohne die technische Gesamtlösung auskommt.

Literatur:

Gibbs, W. Wayt: “Software´s Chronical Crisis”; Scientific American, September 1994

Schinzel B., Meyer L., Winter K.: Komplexität in der Informatik, in Britta Schinzel “Fünf Aufsätze”, Technischer Bericht Nr. 4/96 des IIG der Unversität Freiburg

Schinzel B.: Schnittstellen: Studien zum Verhältnis zwischen Informatik und Gesellschaft, Vieweg, Wiesbaden, 1996.

Schinzel B.: Technikfolgen- und Technikgeneseforschung für die Informatik, in Schinzel B.: Schnittstellen: Studien zum Verhältnis zwischen Informatik und Gesellschaft, Vieweg, Wiesbaden 1996.

Siefkes D.: Formale Methoden und Kleine Systeme, Vieweg, Wiesbaden 1992.

Wohland G.**:** Jenseits von Taylor - Irritation als Methode; in Schinzel B.: Schnittstellen: Studien zum Verhältnis zwischen Informatik und Gesellschaft, Vieweg, Wiesbaden, 1996.