Wege aus der Softwarekrise

Patrick Hamilton

Wege aus der Softwarekrise

Verbesserungen bei der Softwareentwicklung

Springer

Dr. Patrick Hamilton
Am Grundweg 22
64342 Seeheim-Jugenheim

ISBN 978-3-540-72869-6 e-ISBN 978-3-540-72871-9

DOI 10.1007/978-3-540-72871-9

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Einbandgestaltung: KünkelLopka GmbH, Heidelberg

Gedruckt auf säurefreiem Papier

9 8 7 6 5 4 3 2 1

springer.de

Es wäre gut Bücher kaufen,
wenn man die Zeit,
sie zu lesen,
mitkaufen könnte,
aber man verwechselt meistens
den Ankauf der Bücher
mit dem Aneignen ihres Inhalts.
Arthur Schopenhauer (1788–1860)

Inhaltsverzeichnis

1 Prolog

Und wenn das, was du tust, dich nicht weiterbringt, dann tu etwas völlig anderes – statt mehr vom gleichen Falschen!

Paul Watzlawick (1921–2007)

Einmal angenommen, Softwarebau wäre – anders als heute zugrunde gelegt – keine Engineering-Disziplin, sondern wäre stattdessen den psychologienahen Fächern zugeschlagen worden. Einmal angenommen, dieses Fach hätte sich als Folge der seit den frühen 70er Jahren schwelenden Softwarekrise zu einer dort angesiedelten Disziplin gemausert, und einmal angenommen, wir redeten über die Art und Weise, wie die Ergebnisse in Bezug zu ihrer Aufgabenstellung stünden, und nicht so sehr darüber, mit welchen technischen Methoden oder Maschinen man Code implementiert. Würden wir dann in einer anderen Welt leben (soweit es die industrielle IT betrifft)? Könnten möglicherweise Dinge innerhalb der Computerwissenschaften als gelöst betrachtet werden, die sich bis heute als ungelöst, mühevoll und ineffizient darstellen?

Keine Angst, dieses Buch hat es sich nicht zur Aufgabe gesetzt, bewährtes im Softwarebau neu zu definieren. Ausgangspunkt ist hier allerdings eine Betrachtungsweise, deren primärer Fokus nicht techniklastig ist, sondern die, vergleichbar den Kognitionswissenschaften, anstehende Fragen aus einem erweiterten Blickwinkel ergründet und hieraus ihre Schlüsse zieht. Dabei steht im Fokus der Betrachtungen und Überlegungen immer wieder die im industriellen Softwarebau so entscheidende Fragentrias:

Wie wird Softwarebau effizienter?
Wie wird Softwarebau kostengünstiger?
Wie wird die entwickelte Software qualitativ besser?

Viel ist hierzu in den vergangenen Jahrzehnten unternommen und ausprobiert worden. Manches hat sich als gut erwiesen, anderes hat nur für kurze Zeit das Licht der Welt gesehen. Egal welche Methoden und Techniken gerade en vogue sind – diejenige Größe, die sich am wenigsten im Sinne der eigenen Beteiligung verändert hat, ist der Mensch selbst. Auch wenn ihm neuartige Automaten zur Verfügung stehen, wenn Konstruktionshilfen jeglicher Art ihm das Leben beim Bau von Software erleichtern

sollen und eine immer höhere Automatisierung das Ihre dazu beiträgt: Am Ende sind es trotz all der beeindruckenden Technik gerade die Software-Entwickler in ihren verschiedenen Rollen, die durch ihr Wissen, ihr Zusammenspiel und ihre individuellen Vorgehensweisen maßgeblich darüber entscheiden, ob ein Produkt die geforderten Ziele erreicht oder nicht.

Dabei unterliegt industrielle Softwareproduktion ein paar sehr einfachen Grundparadigmen. Die Betriebswirtschaft lehrt, dass ein Produkt in Richtung auf den Markt ein materielles oder immaterielles Objekt sei, das diesem angeboten werde und in der Lage sei, Kundennutzen zu stiften. Mit anderen Worten: Mit einem solchen Produkt möchte die eigene Organisation die Wettbewerbsposition verbessern, bestimmte Marketingziele erreichen, Marktsegmente ausbauen bzw. diese halten – kurz Geld verdienen und damit erfolgreich sein.

Diese Grundsatzziele liegen normalerweise außerhalb des Zeitfensters, in welchem die eigentliche Projektarbeit zur Softwareerstellung stattfindet. Hierdurch werden gerade diese Anliegen oft für die Entwicklungsmannschaften intransparent, weshalb sie leicht in Vergessenheit geraten. Da Softwarebau alles andere als trivial ist, sondern in Hinblick auf die Prozesse, die eingesetzten Werkzeuge und Techniken oder auch die erwarteten Resultate den beteiligten Projektmitarbeitern sehr viel abverlangt, geraten diese Grundparadigmen gerne außer Sichtweite. Dabei ist es längst kaum noch möglich, dass eine geniale Entwicklungsplattform, gepaart mit ein paar guten Ideen sowie dem obligaten Internetanschluss über Nacht zu Wohlstand und Ansehen verhelfen und neue IT-Garagenfirmen erfolgreich das Licht der Welt erblicken.

Dieser Wunschtraum mag bei einigen funktioniert haben oder sich in Ausnahmefällen auch noch so erfüllen, für den Rest von uns heißt es jedoch strategisch ganzheitlich zu planen, ungenutzte Kapazitäten in der eigenen Organisation aufzuspüren, diese zu kanalisieren bzw. fehlerträchtige oder ineffiziente Bereiche zu verbessern. Mit anderen Worten: Statt liebevoller Bastelstuben muss die eigentliche Softwareproduktion immer mehr industrialisiert werden, will sie auch weiterhin den Wünschen der Kunden nach einem entsprechenden Preis-Leistungs-Verhältnis gerecht werden.

Dieser Einsicht folgend werden hier vor allem jene Bereiche analysiert, die Verbesserungspotential in der Herstellung versprechen, oftmals jedoch übersehen oder unterschätzt werden, zugleich aber einen ungenutzten Pool an Ressourcen bereithalten. – Das Hauptaugenmerk dieser Abhandlung wird daher auf einer genaueren Betrachtung der zentralen Rollen im Softwarebau liegen. Dabei werden Skill- und Rollenentwicklung wesentliche Parts zukommen, wobei Wissenserfassung im Sinne des Anforderungsmanagements sowie Wissenstransformation im Sinne von IT-Architektur und Implementierung ein hohes Maß an Aufmerksamkeit erhalten werden.

Bei all dem versteht sich dieses Buch nicht als eine rein akademische Abhandlung, sondern stammt aus vielen Jahren Praxis im Softwarebau sowie dem IT-Beratungsbereich. Es geht in erster Linie um die beteiligten Mitarbeiter und deren qualifizierte Weiterentwicklung. Dies geschieht vor dem Hintergrund, dass viele Publikationen über Produktionsprozesse oder Einzelverfahren den Eindruck vermitteln, dass die eigentliche Workforce in ihren jeweiligen Rollen bereits Teil einer nahezu heilen Welt sei: Mit ihren ohnehin schon vorhandenen Qualifikationen – so der leicht aufkommende Eindruck – brauche diese nur noch auf das jeweils anzuwendende neuartige Vorgehensmodell oder die zu nutzenden Werkzeuge umgeschult zu werden und schon wurde der qualifizierte Umstieg umgesetzt. – Die Wirklichkeit weicht hier ganz offensichtlich von solchen Wunschvorstellungen deutlich ab – und dies insbesondere deshalb, weil die beteiligten Personen mehr als nur ein neues Handbuch oder eine Tagesschulung über einen neuen Softwareprozess bzw. eine entsprechende Methode brauchen, sondern selbst oftmals durch einen mentalen und intellektuellen Veränderungsprozess hindurchgeführt werden müssen.

Wie oft wird Verbesserung der eigenen IT-Organisation in eben der Weise vollzogen, dass die betroffenen Mitarbeiter einen Kurzlehrgang für ein Tool oder ein Verfahren erhalten und man managementseitig im Anschluss daran bessere Ergebnisse erwartet. Dieser Ansatz, der ein wenig an Kochrezepte erinnert, hat mit Sicherheit immer wieder einmal seinen Platz – auch in der Informatik! Aber reicht dies tatsächlich aus, um anspruchsvolle Software zu bauen – und dies mit entsprechend qualifizierten und erfahrenen Mitarbeitern? – Ich fürchte nein! Ein renommierter Architekt wie etwa Norman Forster, der bekannt wurde durch seine genialen Entwürfe, etwa den des Berliner Reichstages oder der Londoner „Gurke“, hat seine Popularität sicherlich nicht deshalb erlangt, weil er mit Hilfe teurer CAD-Systeme, guter Templates oder ausgefallener Werkstoffe geniale Wasserleitungsrohre oder Fassaden perfekt in Bauwerke integrierte. Nein, ganz offensichtlich bedarf es deutlich mehr als nur ingenieurmäßig das Richtige zu tun. Dass dies nicht nur für Bauarchitektur gilt, sondern gleichermaßen auch für den IT-Bereich notwendig ist und z. T. auch schon stattfindet, ist daran zu erkennen, dass große Softwarehäuser, etwa der Gigant Microsoft, dies in ihrer eigenen Binnenorganisation entsprechend berücksichtigen[1].

[1] Hier findet sich – bezogen auf das Jahr 2007 – eine dreiköpfige Führungsmannschaft, die sich zusammensetzt aus der politischen Führung (= klassisches Management), einer kaufmännischen Führung sowie einer technischen Führung, die zum einen Visionär und Impulsgeber ist, zum anderen aber sicherstellt, dass alle Produktentwicklungen in die technische Gesamtvision passen.

Wenn dem so ist, dann bedarf es zweifellos eines Paradigmenwechsels in der Branche. Dieser kann wohl am besten mit demjenigen Wandel verglichen werden, der sich in der vergangenen Dekade im Bereich des Web-Designs vollzogen hat. Waren dort ursprünglich codenahe Kenntnisse gefragt, um z. B. HTML-Tags richtig zu platzieren, die wiederum die Basis der oft stereotypen Klötzchenwelt früherer Web-Seiten waren, so ist heute deutlich anderes gefragt! Anstatt primär die im Hintergrund laufende Technik zu sehen, ist dieser Bereich inzwischen zu einer Domain von Designern geworden. Dabei sind Design und Nutzerinteraktionsplanung zu tragenden Säulen bei der Erstellung solcher Seiten geworden, während die vorhandenen IT-Techniken den für die Realisierung notwendigen technischen Rahmen liefern. Die Erkenntnis hat sich durchgesetzt, dass der wichtigste Kunde solcher Webseiten das menschliche Auge mit seiner nachgeschalteten Wahrnehmung ist. Genau dieses gilt es mit den entsprechend aufbereiteten Informationen sinnesgerecht zu „beliefern"!

Nun kann komplexer Softwarebau nicht ohne weiteres mit Webdesign auf eine Stufe gestellt werden, dennoch gibt es einige Parallelen, die es zu bedenken gilt.

Um es auf den Punkt zu bringen: Wann wird der Tag kommen, an dem standardmäßig im industriellen Softwarebau zentrale IT-Mitarbeiter mit einer Zusatzqualifikation in Gesprächsführung, eher Coaches oder Changemanagern gleich, so mit den Kunden zusammenarbeiten, dass nicht länger IT-technische Dornröschenschlösser mit vielen nachträglichen Erkern und Anbauten entstehen, während der Anwender weiter von märchenhafter Software träumt?

Professionalisierung des Softwarebaus

Erfolg hat nur, wer etwas tut, während er auf den Erfolg wartet.

Thomas Alva Edison (1847–1931), amerikanischer Erfinder

Der Softwarebau hat sich in der Tat erheblich weiterentwickelt, wird aber in den kommenden Jahren noch deutlich auf dem Weg hin zu fabrikmäßiger Produktion zulegen müssen, da es mit Sicherheit kein akzeptabler Zustand ist, wenn um die 50% aller Softwareprojekte[2] nicht im Erwartungshorizont der Auftraggeber enden. Folgt man dem „Chaos-Report" der Standish Group für das Jahr 2006, so wurde in 46% aller registrierten Fälle das Kosten- oder Zeitbudget gesprengt. In der Fachpresse wurde dies als Erfolg gefeiert, da die Quote zwei Jahre zuvor noch stolze 53% betragen

[2] Mit Blick auf das Jahr 2006 wurde die Erreichung der 50%-Marke als großer Erfolg in der Fachpresse gefeiert.

hatte. Ebenso wurde als neuer Rekord gefeiert, dass für denselben Vergleichszeitraum jetzt 35% aller berücksichtigten Softwareprojekte planmäßig in Betrieb gehen konnten, im Gegensatz zu 29% im Berichtszeitraum zwei Jahre zuvor. Wiederum zehn Jahre zuvor, wir schreiben das Jahr 1994, als die Standish Group zum ersten Mal diese Zahlen erhob, belief sich diese Erfolgsquote auf ganze 16%. Und noch eine Auswertung, die zu denken gibt: Im Jahr 1994 musste jedes dritte Softwareprojekt als Totalverlust abgeschrieben werden, inzwischen wird relativ gleichbleibend „nur" noch jedes fünfte bis sechste Projekt komplett versenkt.

Die hier wiedergegebenen Zahlen drücken eine deutliche Verbesserung aus, gleichwohl kann man sich leicht von diesen Quoten blenden lassen! Aber Hand aufs Herz: Würden Sie es als Produzent – egal ob Sie Autos, Fahrräder oder Nähmaschinen herstellen – sonderlich amüsant finden, wenn Sie nur bei jedem zweiten Produkt auf die geplanten Einnahmen kämen und jedes fünfte als Ausschuss aus der Produktion nehmen müssten? – Ich denke, dass kein Industriemanager oder Geschäftsführer im Non-IT-Bereich eine solche Ausschussquote lange akzeptieren würde, wenn ihm sein Job lieb ist!

Auch wenn die Professionalisierung im Softwarebau Fortschritte macht, so kann nicht behauptet werden, dass die von Edgar Dijkstra 1968 erstmals ausgerufene „Softwarekrise" tatsächlich überstanden sei. Aus diesem Grunde ist es notwendig von anderen Branchen zu lernen, die dort gewonnenen Erkenntnisse zu analysieren, wo immer geeignet, sich diese für den IT-Bereich zu eigen zu machen und die Entwicklung mit aller Kraft voranzutreiben. Dabei entsteht der Eindruck, dass sich der Softwarebau derzeit in einer vergleichbaren Situation befindet wie der Automobilbau der 70er Jahre: Während man in Europa und den USA nach den besten Verfahren und Methoden suchte, zog die japanische Automobilindustrie – allen voran der damals fast vom Untergang bedrohte Automobilproduzent Toyota – zum Schrecken aller an der Konkurrenz vorbei und setzte neue Standards in Sachen Effizienz und Qualität (Womack, Jones et al. 1991). Ganzheitliche Produktionsmethoden sowie die optimierte Einbeziehung der wichtigsten Ressourcen, nämlich der Mitarbeiter, bildeten hier die Ausgangsbasis für den Erfolg.

Was aber bedeutet in diesem Kontext Professionalisierung? Folgt man einer kurzen, prägnanten Definition von Scott-Morgan (Scott-Morgan, Hoving et al. 2001), so steht dieser Begriff für Folgendes:

Professionalität = stetiger Fortschritt + keine Überraschungen

Professionalität = Fachwissen + neue Herausforderungen

Dies bezieht sich sowohl auf die Produktseite, wie auch auf die Seite der Akteure. Professionelles Arbeiten des Einzelnen alleine reicht dabei nicht aus. Erst dann, wenn eine Organisation die notwendige Innovationsfähig-

keit aufweist, wird dies dauerhaft zu besseren Produkten und damit verbunden größeren Marktchancen führen.

Erlauben Sie mir daher anhand dieser, zugegebenermaßen sehr einfachen Formeln folgende ketzerische Frage: Leben Sie in einer IT-Organisation, bei der die laufende Produktion bzw. die ausgelieferten Produkte keine regelmäßigen Überraschungen mehr zu bieten haben? – Oder ergeht es Ihnen (wenn Sie ganz ehrlich sind) so wie übrigens immer noch recht vielen IT-Häusern – dass IT Management in der eigenen Organisation gleichzusetzen ist mit dem Bewältigen all der ständig neuen Überraschungen und Hiobsbotschaften.

Auch der zweite Teil der Scott-Morgan'schen Aussage lässt sich durch eine weitere einfache Formel definieren:

Innovationsfähigkeit = Kreativität + Ideenrealisierung

Somit werden aus Business-Sicht einerseits Fachwissen in Verbindung mit Kreativität und andererseits Innovation gepaart mit der tatsächlichen Umsetzung dieser Ideen zu den zentralen Business-Drivern. Dem gegenüber steht ein in der IT-Community recht verbreiteter Ansatz: Er besteht darin, dass primär nach immer neuen Strukturierungsmethoden und technischen Werkzeugen gefahndet wird, um hiermit Software hoch systematisch zu entwickeln. Was aktive Ideenfindung anbelangt, so wird diese vielfach nur diffus erwartet, so als würden die richtigen Ideen an der korrekten Stelle eines Prozesses auf Knopfdruck ins Dasein purzeln. Dementsprechend werden die hierfür notwendigen intellektuellen Handwerkszeuge kaum kommuniziert oder geplant zum Einsatz gebracht. Dabei werden gerade mit deren Hilfe die wesentlichen Voraussetzungen dafür geschaffen, dass ein Unternehmen mit innovativen Produkten und Lösungen schnell und flexibel am Markt agieren kann, um so erfolgreich zu sein zu können.

Während wir durch ingenieurmäßig strukturiertes Vorgehen Software als Konstrukte binärer Logik mit Hilfe solcher Methoden und Modelle entwickeln, werden mehr und mehr Grenzen sichtbar. Diese haben Ähnlichkeiten mit dem Übergang von der klassischen Mechanik zur Quantenmechanik, wo irgendwann einmal schmerzlich klar wurde, dass die schöne und klar umrissene Welt eines Isaac Newton und seiner Fallgesetze eben doch nur in erster Näherung passt und viele Fragen offen lässt. Sich dieser Erkenntnis zu stellen mag für manchen Betroffenen unbequem sein, ist doch jeder Computerchip und jede darauf ablaufende Software eine definierte Abfolge von logischen Schaltelementen bzw. Befehlsketten. Dies ist zweifellos der Fall, berücksichtigt aber nicht, dass die inzwischen ungeheure Menge dieser Schaltungen bzw. Schaltanweisungen superponierte Effekte hervorbringt, die eben nicht mehr ausschließlich klassisch logisch zu verstehen und zu lösen sind, obschon ihnen genau diese Logik zugrunde liegt.

Und so wundert es nicht, wenn Bill Gates davon spricht, wie viel Neuland noch zu betreten sei. In einem Artikel über das, was Datenverarbeitung in den vergangenen 25 Jahren geleistet und hervorgebracht hat, berichtet er (Gates 2007) von den Grundsatzentwicklungen auf dem langen Weg zur Konsolidierung und flächendeckenden Nutzung von Software und weist Perspektiven auf neue Arbeitsfelder aus, wie z. B. die Robotics. Aber ist das Software-Engineering an sich, also diejenige Kunst, die auch bei noch komplexeren IT-technischen Fragestellungen reproduzierbare und qualitativ vorhersagbare Applikationen hervorbringen soll, auch schon bei solch einer Konsolidierung angelangt? Ich meine nein und betrachte die eingangs zitierten Zahlen als einen Indikator dafür.

Zwar versuchen schon längst viele technisch Verantwortliche unter dem Motto „Simplify your IT“ dem Rad in die Speichen zu greifen und eine solche Konsolidierung bzw. Standardisierung in die Wege zu leiten, aber gelingt es ihnen tatsächlich, der rasch wachsenden Komplexität sowie den kontinuierlich neu hinzukommenden Anforderungen entgegenzusteuern? Viel wird investiert, um reproduzierbare Abläufe in möglichst weiten Bereichen der Software-Produktion zu etablieren und somit die Softwareentwickler in einem gewissen Maß zu Fließbandarbeitern zu machen. Ebenso sucht man einem der Hauptfeinde innerhalb der IT-Produktion, nämlich der Komplexität (Hamilton 2006) entgegenzuwirken, indem Standorte geschlossen werden oder einheitliche Plattformprodukte initiiert werden. Was bei all diesen Bemühungen jedoch offen bleibt ist die Frage, ob eine solchermaßen vorgenommene Konsolidierung tatsächlich zum gewünschten Erfolg führt, muss doch die Software von gestern im Sinne von Veränderungsmanagement und die Software von morgen im Sinne von innovativem Planen und Handeln gleichermaßen berücksichtigt werden.

In diesem Sinne möchte ich Sie auf den folgenden Seiten dazu einladen, mich auf der Suche nach Verbesserungspotentialen im Softwarebau von heute und morgen zu begleiten, und dies vorrangig mit Blick auf die eigentlichen Akteure und deren hierfür benötigte Handwerkszeuge.

2 Irrwege und Auswege …

2.1 Softwarebau in der Dauerkrise

Man kann ein Problem nicht mit den gleichen Denkstrukturen lösen, die zu seiner Entstehung beigetragen haben.

Albert Einstein

Es gibt wohl kaum ein Thema, das im Bereich der Softwareentwicklung heftiger, häufiger und auch kontroverser diskutiert wird, als die so genannte Softwarekrise. Existiert eine solche tatsächlich, ist sie längst Historie oder stecken wir immer noch mittendrin?

Die Diskussion über die Frage, ob es diese Krise, von der Dijkstra erstmalig sprach, immer noch gibt, hat spätestens im Jahr 2001 eine ganz neue Facette bekommen, als es über Nacht zu dramatischen Einbrüchen an den Technologiebörsen speziell im Umfeld von Softwarehäusern kam. War Software über Nacht etwa qualitativ deutlich schlechter geworden? Hatte man Trends verschlafen? Oder brach, einer Vulkaneruption gleich, einfach ein totgeglaubtes, vielleicht auch totgeschwiegenes Problem mit lautem Getöse wieder hervor? Was immer die ganze Breite der damaligen Ursachen gewesen sein mag: Tatsache ist, dass in den vorangehenden Jahren die Werte der meisten IT-Firmen bis zu 1.000% an den Börsen gestiegen waren. Manches Unternehmen hatte sich verleiten lassen, mehr auf die eigenen steigenden Aktienkurse zu achten als auf qualitativ hochwertige Entwicklung der eigenen Produkte. Mitten in einer Zeit des Umbruchs, in der erste CASE[3]-Tools angeboten wurden, die durch ihren integrativen Ansatz Planung, Entwurf und Dokumentation nachhaltig verändern sollten, herrschte Euphorie. Man sah sich endlich befreit von den lästigen Schatten der Vergangenheit, vertraute vielfach recht blauäugig auf die Versprechungen der Produktanbieter und stellte oftmals zu spät fest, dass es deutlich mehr bedurfte, als nur Tool-Schulungen, um diese sinnvoll und effizient anzuwenden.

Da es an entsprechend darauf abgestimmten Vorgehensmodellen fehlte, scheiterten viele daran, diese Werkzeuge vernünftig einzusetzen, und so wundert es nicht, dass eine Krise ganz eigener Art – nennen wir sie Tool- und Prozesskrise – zustande kam (Versteegen, Kruchten et al. 2000). Zwar

[3] CASE = Computer Aided Software Engineering

war zu dieser Zeit im deutschen Bereich das V-Modell, andernorts das sog. Wasserfallmodell als Prozessmodell bereits im Einsatz, doch beide Vorgehensmodelle wiesen erhebliche Mängel für den Softwarebau auf, da in sehr statischer Weise Planung, Umsetzung und Qualitätssicherung der neuen Software abgearbeitet werden mussten. Zwar existierten auch schon Spiral- oder Iterationsmodelle (Boehm 1988), doch fehlte es weitgehend an der Durchgängigkeit im produktionstechnischen Sinn. Softwareseitig mauserten sich objektorientierte Methoden und versprachen einen eleganten Ausweg aus der Krise, allerdings tobte ein Methodenstreit, der erst schrittweise beigelegt wurde.

Sucht man nach den damaligen Ursachen hierfür, so spielte die Anforderungserfassung sowie ihre technische Umsetzung eine wesentliche Rolle, da Anforderungen unklar formuliert bzw. nicht verstanden oder falsch bzw. unvollständig erfasst wurden und in Folge dessen instabil waren. Die Folge war vielfach eine falsche oder unvollständige Umsetzung, die erheblichen Einfluss auf die Realisierung der angestrebten technischen Lösungen hatte. Überdies kämpfte man in der Branche mit den immer wieder auftretenden Technologiesprüngen sowie mit dem Mangel an qualifizierten Mitarbeitern.

Dies sind vorrangige Gründe für die Probleme, die man um die Jahrtausendwende im industriellen Softwarebau hatte. – Aber sind es nicht genau dieselben Probleme, mit denen wir noch heute zu kämpfen haben, obwohl viele „Best Practices“ bzw. neue Entwicklungstools eingeführt wurden, man viel in Qualitätssicherung investiert hat und vielfach große Anstrengungen unternommen wurden, um Softwarebau zu prozessualisieren?

Obschon moderner Softwarebau auf mehrere Jahrzehnte an Erfahrung blickt, bewegt er sich immer noch nicht auf einem einfachen Weg, sondern gerät immer wieder in neue Krisen. Mit wachsender Komplexität der neuen Produkte sowie neu entstehenden technischen Hilfsmitteln verlagerte sich die eigentliche Problemzone zunächst Richtung Projektmanagement und dann weiter Richtung Anforderungserfassung. Mehr und mehr kamen in den 1990er Jahren Faktoren ins Spiel, die nur im Zusammenwirken der unterschiedlichen Beteiligten als aufeinander eingespieltes Team zu bewältigen waren, wofür geeignete technische Austauschplattformen erforderlich wurden.

Was den immer wieder neu auftretenden Krisenzeiten gemeinsam ist, liegt interessanterweise nicht primär im Bereich technischer Ursachen, sondern hat zu tun mit den komplexen sozialen und organisatorischen Rahmenbedingungen. Dies zumindest belegen Studien der ETH in Zürich (Kuhnt and Huber 2001)[4].

[4] Am Institut für Informatik der Universität Zürich setzt man sich mit Fragen des IT-Projektmanagements auseinander und untersucht dort die Gruppenprozesse in IT-Projekten.

2.1.1 In den Sümpfen des mystischen Mann-Monats

Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest.

Frederick Brooks

Bei all dem bewegen wir uns in einem Kontext, der keineswegs neu ist. Seitdem Frederick Brooks im Jahr 1975 seine IT-Essays über den sagenumwobenen „Mann-Monat“ (Brooks 1995) erstmals veröffentlichte, hat sich technologisch betrachtet sehr vieles verändert. Brooks, der viele Jahre als IT-Professor und zugleich bei IBM für die Hardware-Entwicklung des „Systems 360“ verantwortlich war, untersuchte in diesem Werk die Eigendynamik großer Softwareprojekte. Wir begegnen dort den Monstern der IT, nämlich nicht eingehaltenen Terminen, gesprengten Budgets sowie fehlerhaften Produkten. Aspekten also, die sich im ganz normalen Entwicklungsalltag zunächst als harmlose Zeitgenossen ankündigen und dann auf einmal, um in der Sprache von Brooks zu bleiben, sich aus dem Nichts in furchterregende Werwölfe verwandeln, für deren Erlegung man die berühmte silberne Kugel braucht, nämlich diejenige Wunderwaffe, um diesen Ungetümen den Garaus zu machen.

Klingt dies nicht mehr als dreißig Jahre später immer noch äußerst aktuell? Haben nicht über all die Jahre hinweg immer wieder wissenschaftliche Institute oder Entwicklungshäuser den Sieg, die allein rettende Wunderwaffe aufgrund technischer Neuerungen ausgerufen, und ist es nicht ein Faktum, dass seit den frühen 70er Jahren des vergangenen Jahrhunderts diese Krise laut Brooks oder Dijkstra existiert und keine Entwicklung sie wirklich aufgelöst hat?

Dabei wäre es sicherlich ungerecht all den vielen genialen und weniger genialen Köpfen, die die IT seither massiv weitergebracht haben, die gelbe Karte zu zeigen. Hatte Brooks vielleicht doch Recht, als er in seinem Werk postulierte, dass er keinen überraschenden Erfolg erwarten könne und binnen der folgenden 10 Jahre nicht mit wirklichen Durchbrüchen in Bezug auf die Erledigung dieses Ungetüms zu rechnen sei? – Ja, ist es nicht sogar noch viel schlimmer gekommen? Ist nicht inzwischen über ein Vierteljahrhundert verstrichen, ohne dass es zu wirklichen Erfolgen auf breiter Front kam? Wie löst sich die Problematik der menschlichen Fehler in der Softwareentwicklung – noch dazu, wo Werkzeuge, Verfahren und Applikationen immer komplexer werden? – Reicht es aus, Menschen mit genial durchdachten Checklisten und Prozessvorgaben ans Werk zu schicken, da-

mit sie auf keinen Fall einen der vielen Teilschritte im Softwarebau übersehen oder gar vergessen? Ist der Mensch in dieser wohl kompliziertesten aller von im gehandhabten Technologien einfach am Rande seiner intellektuellen Kapazitäten? – Oder könnte es sein, dass wir aus lauter Technik- und Prozessverliebtheit wesentliche Aspekte im Softwarebau unzureichend betrachtet haben?

Lassen Sie uns hierzu den Ausführungen von Brooks folgen und die Schwerpunkte seiner Ursachenforschung von damals zusammenfassen:

Änderbarkeit: Im Gegensatz zu anderen Industriegütern, die seiner Meinung nach nur einer geringen Veränderungsrate unterliegen, ist Software einer hohen Änderungsquote unterworfen, insbesondere da keine Materialkosten anfallen.

Unsichtbarkeit: Zur der Zeit, als Brooks diese Aussagen machte, wurde Software noch nicht visualisiert. Entsprechend fehlte es an graphischen Werkzeugen und Diagrammformen, also all dem, was heute moderne Tools und Modellwelten zu bieten haben. Die Folge: Abstraktion führt sehr viel leichter zu Fehlern.

Komplexität: Da jede Software auf ihre Weise ein Unikat ist, bedeutet Softwareerweiterung nicht notwendigerweise eine Wiederholung bestehender Elemente, sondern die Schaffung neuer. Da Komplexität ein zentrales Problem im Softwarebau ist und diese überproportional mit der Größe eines Programms zunimmt, stellen sich hierdurch Zusatzprobleme ein, etwa Sicherheitslücken aufgrund nicht dargestellter Zustände.

Konformität: Programme müssen den Anforderungen der Benutzer entsprechen, diese aber sind oftmals willkürlich! Ebenso muss Software sich an gegebene Situationen anpassen können, bzw. es werden extrem hohe Anforderungen an solche Systeme gestellt. Diese wiederum bewirken eine Erhöhung der Komplexität.

Folgt man diesen Problemkreisen im Softwarebau, so lässt sich im Hinblick auf heute sehr wohl erkennen, wo überall bereits Baulücken geschlossen wurden, seitdem Brooks diese Themenfelder postulierte. Inzwischen sind visuelle Tools, das Modellieren von Use Cases oder auch prozessgetriebenes Arbeiten in unterschiedlichen Ausprägungen gangbare Möglichkeiten, um Software zu bauen.

2.1.2 Fehler ohne Verjährungsfristen

Operative Hektik ist ein Zeichen geistiger Windstille.

Joseph Schmidt, Management-Trainer

Softwarebau ist weit mehr als nur das geschickte Platzieren von Bits und Bytes, das Definieren von Objekten und Strukturen bzw. das Implementieren von Geschäftsmodellen und Prozessen. Hat man in den Planungsphasen vorwiegend diese Bereiche im Auge, so darf es nicht verwundern, wenn eines der nachfolgenden Ergebnisse zutage tritt.

Zufallsdesign: (engl. Design by Default) Während der Entwicklung hat man sich auf die eingesetzte Technik und die geforderten Funktionalitäten konzentriert. Eher zufällig sind dabei die Gesamtstruktur wie auch die Schnittstellen zum Anwender entstanden. Eine gezielte Planung in diesen Bereichen fand nicht oder viel zu spät statt. – Die Folge: am Ende hat man ein Produkt, das sehr stark von den eingesetzten Techniken geprägt ist. Benutzerfreundlichkeit ist nicht unbedingt die Stärke einer solchen Anwendung.

Nachahmungsdesign: (engl. Design by Mimicry) Dies ist nicht das erste Softwareprojekt einer Organisation, und so kopiert man Fragmente von eigenen Produkten oder man kupfert von Fremdprodukten so viel wie möglich ab. Dies geschieht in der Hoffnung, so die eigenen Entwicklungsaufwände zu begrenzen. Planungen über gezielte Wiederverwendbarkeit oder von echten Produktfamilien[5] fanden nicht oder zu spät statt. Die Folge: Der Pflegeaufwand für diesen „Zoo“ an ähnlichen Programmen ist immens, da jedes Produkt, trotz seiner scheinbaren Ähnlichkeit, individuell angepasst bzw. gepflegt werden muss.

Schicksalsdesign: Ohne einen zentralen „Mastermind“, der ein Gesamtkonzept verfolgt und Impulsgeber für das Gesamtergebnis ist, sitzen viele Spezialisten an ihrem spezifischen Bereich und liefern hierfür entsprechende Lösungen. Auch wenn technische Abstimmungen zu anderen Fachbereichen stattfanden und auch am Ende ein funktionierendes Produkt entstand, so ist das Gesamtprodukt eher mit einer bunten Blumenwiese unterschiedlicher konzeptioneller und technischer Ausprägungen mit vielfältigen Redundanzen zu vergleichen, dem zugrundeliegende gemeinsame

[5] Produktfamilien bilden sich aus einer Gruppe von Produkten, die in definierten Bereichen auf gleiche Komponenten zurückgreifen (= Commonalities), während sie in anderen definierten Bereichen voneinander abweichen (= Variabilities). Typischerweise werden Produktfamilien dazu verwandt, unterschiedliche Teilbereiche eines größeren Gesamtmarktes individuell zu bedienen.

Konzepte fehlen. Die Folge: auch wenn man es vor dem Kunden verheimlichen kann – aber ein Blick in das Innere dieser Konstruktion lässt nur noch einen Satz über die Lippen kommen: „Hier wurde liebevoll, aber nicht konsequent strukturiert gearbeitet, schlimmstenfalls gebastelt".

Können wir uns solche Arten ineffizienten Softwarebaus in einer Zeit, wo es um industrielle Qualität und Zertifizierungen, aber auch hohen Wettbewerbsdruck geht, noch leisten? – Mehr noch: Wie immer die in Ihrer Organisation entwickelte Software aussehen mag, Fakt ist, dass es intrinsische Eigenschaften gibt, die im Lebenszyklus des Produktes zu erheblichen Risiken und auch Folgekosten führen können. Hier einige Beispiele:

- Software ist nicht nur als Problemlöser zu betrachten, sondern stellt aufgrund seiner inhärenten Eigenschaften ein erhebliches Problem und somit auch Risiko da. – Kennen Sie die individuellen Risiken und deren Ursachen?
- Software wird für ein einfach zu modifizierendes Produkt gehalten. Es trifft zwar zu, dass Programmcode, ohne Materialkosten zu verursachen, geändert werden kann, die zielgerechte und korrekte Änderung ist wegen der dabei möglichen Verletzung der gewünschten inneren Wirkzusammenhänge oder wegen der möglichen Hinzufügung neuer, nicht beabsichtigter innerer Wirkzusammenhänge jedoch extrem teuer. – Sind diese Wirkzusammenhänge in Ihrer Organisation bekannt und haben Sie effektive Wege damit umzugehen?
- Software wird oftmals für ein preiswert zu erstellendes Produkt gehalten. Preiswert ist lediglich die Vervielfältigung eines einmal entwickelten Systems, die Ersterstellung ist allerdings sehr personal- und somit kostenintensiv. – Was tun Sie dafür, dass Ihre Mitarbeiter maximal effizient arbeiten können?
- Software gewinnt im Laufe der Zeit eine weitgehende Eigenständigkeit dadurch, dass die meisten Anwender die Detailkenntnisse der durch sie repräsentierten Problemlösungsverfahren verlieren[6]. Software wird somit zum eigentlichen Wissensspeicher in der jeweiligen Organisation oder auch fachlichen Domain – wobei dieses Wissen oftmals nur noch stark verschlüsselt und somit nicht mehr direkt verstehbar vorliegt. – Haben Sie die Wissensproblematik in Ihrer Organisation im Griff oder sind Sie zum Friedhofsgärtner Ihrer eigenen Datengräber avanciert?
- Auch wenn viele Dokumente über die eigene Software existieren, so driften Wirklichkeit (= was tatsächlich kodiert wurde) und Fiktion (= was tatsächlich dokumentiert wurde) oftmals auseinander. – Wie halten Sie in der eigenen Organisation Sein und Schein beieinander?

[6] Ein ganz simples Alltagsbeispiel: Können Sie noch per Hand Wurzeln ziehen – oder macht das Ihr Taschenrechner bzw. Computer?

- Software wird fälschlicherweise als ein nicht verschleißendes und damit nicht alterndes Produkt angesehen. Die fortlaufende Korrektur und die immer wieder notwendig werdenden Änderungen und Erweiterungen – zur Vervollkommnung des durch sie repräsentierten Problemlösungsverfahrens – führen zum „Einbau" nichtbeabsichtigter Effekte, zu strukturellen Verwaschungen und damit zum Altern. – Wie sehen die „Anti-Aging"-Pillen für Ihre Software-Produkte aus – oder leben Sie nach dem St. Florians-Prinzip?

Trotz aller eingesetzten Techniken und Methoden: Wenn es auf diese und andere Fragen keine oder nur unzureichende Antworten gibt, so sind genau diese Themen es, die im Laufe der Jahre in erheblichem Maß zum Risiko und zur Kostenfalle werden. – Wenn also diese Aspekte zu demjenigen Stoff gehören, mit dessen Konsequenzen wir uns täglich auseinandersetzen müssen, so ist zu fragen, wie diesen rechtzeitig begegnet werden kann. – Auch wenn es in der Branche immer noch recht beliebt ist, auftretende Probleme erst dann abzuarbeiten, wenn es brennt, so ist anzumerken, dass grundsätzliche und rechtzeitige Antworten sehr viel zielführender sind als nur die Beseitigung aktueller Störfälle. Im Sinne eines TQM[7] oder anderer qualitätsgetriebener Vorgehensweisen geht es darum, Probleme an der Wurzel zu lösen. Auch wenn mancher Manager Problemlösung und Ziele in einen Topf wirft und folglich aus der Lösung von Problemen Zielerreichungsaufgaben definiert (Radatz 2004, S. 132ff.), an welchen dann wiederum Zusatzvergütung – vielleicht auch die eigene – hängt, so muss man sich Folgendes klar machen:

Indem Sie Probleme abarbeiten, beseitigen Sie sichtbare Defizite und gelangen somit auf den Null-Stand – aber auch keinen Deut weiter! Dies hat zur Folge, dass Sie – gerade im Softwarebau – ein paar Fehler weniger haben werden (hoffentlich!), jedoch keine Innovation. Mit dererlei Ansätzen betreiben Sie Symptom-, aber keine Ursachenbekämpfung!

2.1.3 Handeln mit Kurzsichtigkeit

Wir leben in einer Zeit vollkommener Mittel und verworrener Ziele.

Albert Einstein (1879–1955, dt.-amerik. Wissenschaftler)

Softwarebau isoliert von der jeweiligen Organisation und ihren Entscheidungsträgern zu sehen ist blauäugig. Aus diesem Grunde erscheint es notwendig einen Teil der Krise außerhalb der eigentlichen IT zu suchen, und zwar in den Köpfen der Verantwortlichen bzw. Beteiligten. Grundsätzli-

[7] Total Quality Management