Softwarewartung

Grundlagen, Management und Wartungstechniken

Bearbeitet von
Christoph Bommer, Markus Spindler, Volkert Barr

1. Auflage 2008. Broschüren im Ordner. 328 S.
ISBN 978 3 89864 482 2
Format (B x L): 16,5 x 24 cm

1 Einleitung

Genauso wie Menschen älter werden, so wird auch Software älter. Dieser Prozess ist für beide unumgänglich und kann nicht verhindert werden. Wir können diesen Vorgang aber besser verstehen lernen und sogar Schritte unternehmen, um verschiedene Auswirkungen einzuschränken oder zu verlangsamen. Im besten Falle können wir sogar vereinzelte Probleme, die das Älterwerden mit sich bringt, rückgängig machen.

Software-Alterung ist kein neues Phänomen, aber wegen der zunehmenden wirtschaftlichen Bedeutung steigt das Interesse an diesem Thema stetig. Die Software ist in der Zwischenzeit bei vielen Firmen ein gewichtiges Stück *Kapital* geworden und erfüllt entsprechend wichtige Aufgaben in unserer Gesellschaft. Das Älterwerden dieser Programme bedroht aber die weitere Entwicklung der davon abhängigen Systeme. Daher ist es umso wichtiger, sich mit diesem Thema intensiver auseinanderzusetzen.

Wer heute neue Software auf der »grünen Wiese« entwickelt, hat für obige Worte vielleicht nur ein mitleidvolles Lächeln übrig und mag denken, dass sein System davor gewappnet ist. Schließlich hat er ja die neueste Programmiersprache verwendet, eine astreine Architektur hingelegt und lehrbuchmäßig die Design Patterns eingesetzt – was soll da noch schiefgehen? Begegnet man heute älteren Menschen, so kann man sich auch kaum vorstellen, was sie in jungen Jahren alles geleistet haben. Vielleicht war der ältere Herr in unserer Nachbarschaft einmal ein hervorragender Sportler und die ältere Dame von Gegenüber eine wunderschöne und begehrte Schauspielerin. Daher können wir nur all denjenigen, die meinen, ihr neues System sei vor dem Älterwerden gewappnet, zu bedenken geben: »So wie wir heute vielleicht über ein Cobol-Programm lächeln mögen, so werden unsere Kinder eines Tages über Java-Programme lächeln.«

1.1 Demografie in der Software

Entwickelte man früher eine Applikation, schöpfte man damit meist die Möglichkeiten der verwendeten Hardware voll und ganz aus. Kaum stand die nächste Generation von Hardware zur Verfügung, die natürlich deutlich schneller war,

wurde die Applikation neu geschrieben und die neue Hardware wieder voll ausgenutzt. Trotz der rasanten Entwicklung im Hardwareumfeld, schöpften die neuen Programme die Ressourcen immer wieder schnell aus. Die Kontinuität der Plattform (Hardware und Betriebssystem) war durch Wechsel der Prozessoren, Datenbusbreite, Filesysteme, Linkformate usw. häufig nicht gegeben, was das Wachstum großer Softwaresysteme hemmte. Aufseiten der Programmierumgebungen fehlten häufig umfangreiche Bibliotheken, sodass man sich nicht selten mit dem Entwickeln von verketteten Listen und dergleichen aufhielt, oder sie waren proprietär und wurden plötzlich, unter neuen Plattformen, nicht mehr angeboten. Auch die Softwareentwicklungsumgebungen unterstützten die Entwicklung großer Softwaresysteme nur begrenzt.

Aus diesen Gründen war es nur beschränkt möglich, sehr große und komplexe Systeme zu entwickeln, die typischerweise über Jahre und Jahrzehnte hinweg wuchsen. Denn dazu braucht es folgende zwei Voraussetzungen:

- eine performante Plattform (Hardware/Betriebssystem), die Kontinuität erlaubt, und
- Softwareentwicklungswerkzeuge, die die Entwicklung großer Systeme unterstützen.

Im Laufe der Zeit, mit den stetigen neuen Errungenschaften, verbesserten sich die Kontinuität der Plattformen sowie die Entwicklungsumgebungen und erlaubten es schließlich, immer größere und komplexere Probleme mittels Software lösen zu können.

Dieses Wachstum großer Applikationen kann man an den Beispielen der Betriebssysteme Windows und Red Hat Linux gut erkennen (s. Abb. 1–1).

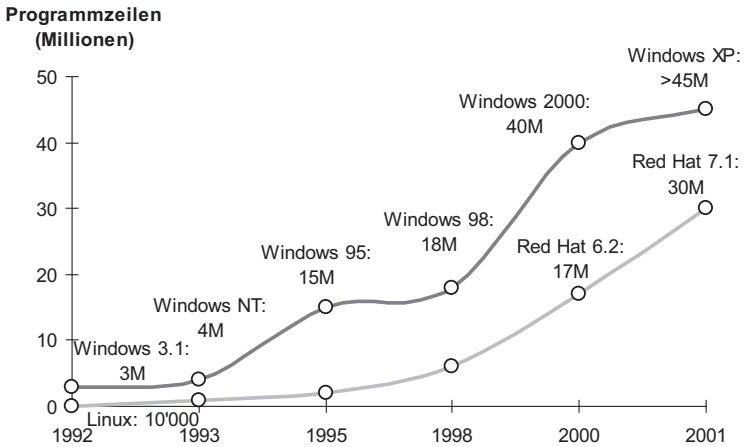

Abb. 1–1 *Wachstum von Betriebssystemen*

Abbildung 1–1 zeigt deutlich, wie schnell die verschiedenen Betriebssysteme gewachsen sind. Um sich diese riesigen Zahlen besser vorstellen zu können, kann man mit einem rund 1,5 Meter hohen Papierstapel pro einer Million ausgedruckter Codezeilen rechnen. Damit ergäbe dies zum Beispiel für Windows XP ungefähr 68 Meter, was etwa einem Haus mit 22 Stockwerken entspricht.

In große Systeme, die über viele Jahre und zum Teil Jahrzehnte gewachsen sind, wurde enorm viel investiert, sie können nicht so ohne Weiteres ersetzt werden. Viel eher geht es darum, diese Investitionen durch eine entsprechende Wartung zu schützen. In Deutschland wurde deshalb Anfang 2005 eine Software-Initiative [SWI05] gegründet, mit dem Ziel, Wege und Strategien aufzuzeigen, wie diese kritische Software erhalten und modernisiert werden kann. Man will Unternehmen und öffentlichen Verwaltungen dabei helfen, die über zwei Billionen Euro zu retten, die in den vergangenen Jahrzehnten in Bestandsapplikationen investiert wurden. Gemäß dem Vorsitzenden der Software Initiative Deutschland, Helmut Blank, sind 70 % aller für die Volkswirtschaft und das öffentliche Leben relevanten Applikationen immer noch in Sprachen wie Cobol, PL/1 oder Assembler geschrieben und laufen auf Mainframes. Dabei machen die Cobol-Applikationen den Löwenanteil aus, auf denen täglich etwa 30 Milliarden Transaktionen abgewickelt werden, die auf rund 45.000 Großrechnern laufen. Für die Wartung der Oldie-Software werden allein in Deutschland rund zehn Milliarden Euro jährlich aufgewendet. Die Umstellung der etwa 240 Milliarden Cobol-Programmzeilen – ohne PL/1 und Assembler – auf eine moderne und damit wartungsfreundlichere Programmiersprache würde gemäß der Software Initiative Deutschland geschätzte 25 Euro pro Codezeile, in Summe rund sechs Billionen Euro kosten und damit den Kostenrahmen der Volkswirtschaften und der öffentlichen Haushalte sprengen. Hinzu kommt, dass die Überführung Jahrzehnte benötigen würde und die heute modernen Umgebungen bis dahin ebenfalls schon wieder veraltet wären. Umso wichtiger ist eine gezielte Softwarewartung (alle Zahlen aus [SWI05]).

Ein weiterer Hinweis, dass die Softwarewartung weiter zunehmen wird, liefert Capers Jones [Jon96]. Laut ihm zeigt sich oft ein kritisches Phänomen einer Industrie dann, wenn sie 50 wird, denn in diesem Stadium braucht es mehr Leute, die ein Produkt pflegen, als solche, die es herstellen. So zum Beispiel geschehen, als die Autoindustrie 50 Jahre alt wurde. Damals waren in den Vereinigten Staaten von Amerika viel mehr Leute mit der Reparatur von Autos beschäftigt als mit dem Herstellen neuer Autos in den Fabriken. Ein ähnlicher Trend zeichnet sich in der inzwischen ebenfalls 50-jährigen Softwareindustrie ab. Tabelle 1–1 zeigt die ungefähre Anzahl Programmierer, die in den Bereichen Erstentwicklung, Erweiterungen und Wartung seit 1950 bis 2020 tätig sind. Wie man sieht, sind seit 1990 erstmals mehr Leute mit Erweiterungen und Wartung bestehender Systeme beschäftigt als mit der Erstellung neuer Systeme. Dieser Trend wird zusätzlich durch die Effekte der höheren Produktivität der Programmierung sowie der größeren

Verbreitung von Software unterstützt. Gemäß Capers Jones sind die Daten zum Teil empirisch, zum Teil durch Extrapolation entstanden.

Jahr	Programmierer Erstentwicklung	Programmierer Erweiterungen	Programmierer Wartung	Total
1950	90	3	7	100
1960	8.500	500	1.000	10.000
1970	65.000	15.000	20.000	100.000
1980	1.200.000	600.000	200.000	2.000.000
1990	3.000.000	3.000.000	1.000.000	7.000.000
2000	4.000.000	4.500.000	1.500.000	10.000.000
2010	5.000.000	7.000.000	2.000.000	14.000.000
2020	7.000.000	11.000.000	3.000.000	21.000.000

Tab. 1–1 *Anzahl Programmierer in der Wartung*

All diese Zahlen belegen eindrucksvoll, wie wichtig es ist, sich mit dem Thema der alternden Softwaresysteme auseinanderzusetzen. Selbst wenn die Zuverlässigkeit der Daten beider Beispiele nicht so hoch sein sollte, der Trend ist sicher richtig.

Dank der oben beschriebenen rasanten Entwicklung der letzten Jahrzehnte entstanden zum einen immer beständigere Plattformen und zum anderen ist es möglich geworden, immer komplexere Probleme mittels Software zu lösen. Beides zusammen, die stabilen Plattformen als Investitionsschutz sowie die Möglichkeit, immer komplexere Probleme mit Software abzubilden, hat zu wichtigen und großen Systemen geführt. Ohne diese Systeme würde unsere moderne Gesellschaft, wie wir nachfolgend noch sehen werden, nicht mehr funktionieren. Heute gilt es daher, diese zum Teil in die Jahre gekommenen Programme zu beherrschen und zu warten. Dass dies nicht immer so einfach ist und sogar zu Pannen führen kann, wollen wir uns als Nächstes anschauen.

1.2 Größe als Herausforderung

Die Größe von Softwaresystemen führt nicht nur zu schützenswerten Investitionen, sondern beinhaltet auch Risiken und Abhängigkeiten. Denn allein die Größe der Software bringt unweigerlich eine gewisse Anzahl unentdeckter Fehler mit sich. Gehen wir aber der Reihe nach vor. Aus der Industrie kennen wir Durchschnittswerte von Fehlerraten, deren Auswirkung auf unsere Software wir uns nun anschauen wollen [Tah04]. Dies ist zwar keine exakte Wissenschaft, aber eine fundierte Schätzung dürfte für unsere Zwecke allemal ausreichen. Zunächst findet man während der Entwicklung 30 bis 50 Fehler pro 1.000 Lines of Code

mittels Tests. Trotz all dieser Tests verbleiben aber Restfehler. Restfehler sind unentdeckte Fehler, die in der ausgelieferten Software immer noch vorhanden sind. Bei einer guten Organisation der Softwareindustrie kann man mit eins bis drei Restfehlern pro 1.000 Lines of Code rechnen. Tabelle 1–2 gibt dazu einen Überblick.

Programmumfang in Lines of Code (LOC)	Fehler während der Entwicklung	Restfehler	Schwerwiegende Restfehler
1.000	30–50	1–3	0,1–0,3
50.000	1.500–2.500	50–150	5–15
100.000	3.000–5.000	100–300	10–30
500.000	15.000–25.000	500–1.500	50–150
1.000.000	30.000–50.000	1.000–3.000	100–300

Tab. 1–2 *Fehlerverteilung*

Natürlich ist nicht jeder Restfehler ernsthafter Natur. Schließlich muss ein Tippfehler, der sich in einem Text eines Programms eingeschlichen hat, nicht gleich zu einem Absturz des Systems führen. In der Industrie rechnet man damit, dass rund 10 % der Restfehler für das System ernsthafte Konsequenzen haben und so entweder zu einem Absturz oder einer Verklemmung *(deadlock)* führen können (alle Zahlen aus [Tah04]).

Was heißt das nun für unsere Programme? Dies bedeutet also für die Software eines Handys (s. Abb. 1–2), die 200.000 Codezeilen umfasst, bereits 200 bis 600 Restfehler und 20 bis 60 schwerwiegende Restfehler. Nun ist ein schwerwiegender Restfehler in einem Handy das eine und in einem Spaceshuttle das andere. Daher trifft man bei der Entwicklung wichtiger Systeme mit etlichen zusätzlichen Maßnahmen Vorkehrungen, sodass man die Fehlerraten nochmals um den Faktor zehn verringern kann. Das heißt für die besonders sicher geltende Software des Spaceshuttles, die drei Millionen Codezeilen umfasst und »lediglich« 0,1 Restfehler pro 1.000 Codezeilen aufweist, dass immer noch stolze 300 unentdeckte Fehler im Programm verblieben sind und davon 30 als schwerwiegend eingestuft werden können!

Die Tatsache, dass Programme nach deren Auslieferung immer noch mit Restfehlern und damit auch mit schwerwiegenden Restfehlern behaftet sind, führt immer wieder zu größeren Computerpannen, wie man jeweils aus der Presse erfahren kann. Dies überrascht bei neuen Programmen sicherlich niemanden mehr. Nicht, dass das so sein sollte, aber man hat sich offenbar daran gewöhnt.

Etwas mehr überrascht ist man vielleicht, dass große Pannen auch bei langjährigen und bewährten Systemen auftreten können. Nachfolgende Meldungen aus Zeitung und News greifen bewusst zwei Beispiele von Computerpannen auf, die auf unsachgemäßen Umgang mit alten Computersystemen zurückzuführen sind.

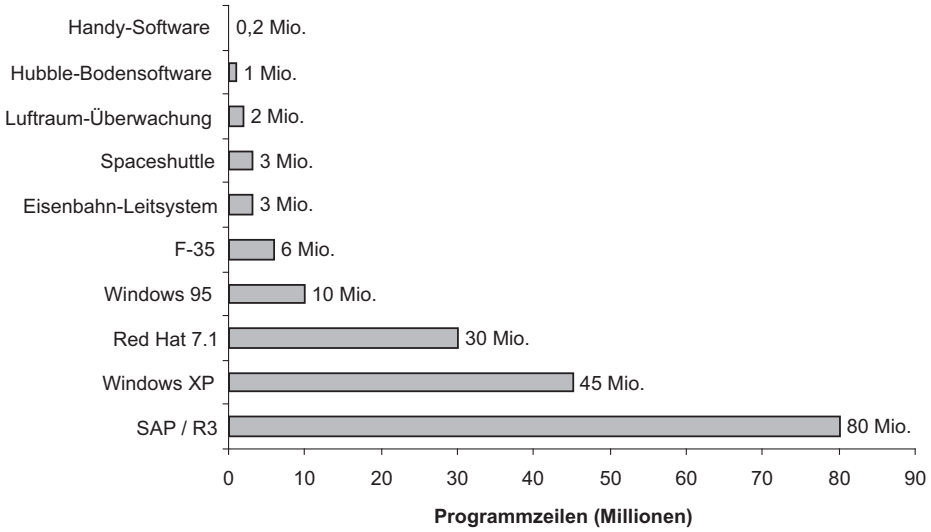

Abb. 1–2 *Größe von diversen Systemen*

Im Juni 2004 kam es auf dem Flughafen London-Heathrow wegen einer Computerpanne zum Chaos (s. Kasten »Panne: Britischer Flugverkehr lahm gelegt«). Der Ausfall des computergesteuerten Flugkontrollsystems führte dazu, dass auf allen britischen Flughäfen die meisten Starts auf unbestimmte Zeit verschoben werden mussten. Zehntausende Reisende mussten mehrstündige Verspätungen hinnehmen. Was war geschehen?

»Panne: Britischer Flugverkehr lahm gelegt«

(diepresse.com 03.06.2004)

Ein Test mit einer verbesserten Softwareversion führte zum Ausfall der computergesteuerten britischen Flugüberwachung. Auf den Flughäfen herrschte zeitweise Chaos.

Eine Computerpanne im Zentralcomputer der privatisierten Nationalen Flugüberwachung führte zum Chaos im britischen Flugverkehr. Gegen 7 Uhr MESZ fiel am Donnerstag das komplette computergesteuerte Flugkontrollsystem aus, auf allen britischen Flughäfen mussten die meisten Starts auf unbestimmte Zeit verschoben werden. Nach rund einer Stunden konnte das Computerproblem behoben werden, der Verkehr kam am Vormittag erst langsam wieder in Gang.

Zehntausende Reisende mussten mehrstündige Verspätungen hinnehmen. London-Heathrow, der größte Flughafen Europas, meldete laut APA »ernsthafte Probleme«. Das System stürzte in Folge eines nächtlichen Tests mit einer verbesserten Softwareversion ab. Die Flugsicherheit sei zu keinem Zeitpunkt beeinträchtigt gewesen, versicherte die Flugüberwachung National Air Traffic Services (NATS). Es wurden alle Starts vorerst abgesagt, damit die Fluglotsen sich vorrangig um die bereits in der Luft befindenden Maschinen kümmern können.

Der Fehler ist in einem Teil des Systems aufgetreten, der bereits 30 Jahre alt ist und erst 2011 ersetzt werden soll. Dabei wurde eigentlich in britischen Medien durchweg von »einem der modernsten Computersysteme der Flugüberwachung weltweit« berichtet. Ein ähnlicher Fehler trat aber bereits vor zwei und vor vier Jahren auf, wurde im Laufe des Tages bekannt.

Die Panne sei ein »weiteres Beispiel für die Inkompetenz der Regierung«, machte der konservative Oppositionsführer Michael Howard das Team von Premierminister Tony Blair laut BBC für das Chaos verantwortlich. Die Nationale Flugüberwachung wurde in den vergangenen Jahren gegen erbitterten Widerstand auch aus den Reihen der Labour-Partei teilprivatisiert. Flugsicherheit und Geldverdienen passten nicht zusammen, lautete das Argument der Gegner. (ito)

Ein Vorfall, den direkt Betroffene sicher bemerkt, die meisten anderen aber kaum wahrgenommen haben. Der Clou ist, dass es offensichtlich nicht einmal der einzige Vorfall mit diesem 30-jährigen System war, sondern immer wieder Probleme aufgetreten sind. In diesem Falle gilt hier als Ursache das nächtliche Update der aktuellen Software, das zum Ausfall des Systems geführt hat. Provokativ könnte man fragen, ob man nach 30 Jahren Erfahrung immer noch nicht in der Lage ist, dieses System fehlerfrei zu aktualisieren. Da wir aber davon ausgehen, dass Spezialisten am Werk waren, ist der Umgang mit einem solch alten System offensichtlich nicht so einfach, wie man im ersten Moment meinen könnte. Folgende Fragen dazu lassen uns das eine oder andere Problem in diesem Umfeld erahnen:

- Gab es Entwickler, die das System noch wirklich kannten?
- Existieren Dokumentationen, die das System, die Architektur und Umsysteme beschreiben?
- Welche Tests wurden im Labor durchgeführt?
- Welche Komplexität wies das System nach 30-jähriger Wartungsarbeit auf?
- War die Hardware noch auf einem aktuellen Stand?
- Wurde das System ursprünglich für die eingesetzte Hardware entwickelt?

Die Antworten kennen wir in diesem Fall nicht, sie zeigen aber mögliche Problemfelder im Umgang mit älterer Software auf. Mit 30 Jahren, und das müssen wir eingestehen, kommt eine Softwarelösung tatsächlich ins hohe Alter. Die Softwarebranche ist derart dynamisch, dass Softwareapplikationen schon nach wenigen Jahren alt erscheinen. Dazu ein kleiner Vergleich: Bekanntlich entspricht gemäß einer Faustregel ein Hundejahr sieben Menschenjahren. Wenn wir als These unterstellen, dass es sich bei Software ähnlich verhält, hätten wir es hier mit einem ungefähr 210-jährigen System zu tun.

Wie man erahnen kann, birgt der Umgang mit alternden Systemen etliche Herausforderungen. Wenden wir uns einem zweiten Fall einer schwerwiegenden Computerpanne zu, dem Absturz der Ariane-5-Trägerrakete. 39 Sekunden nach dem Start der Rakete zerstörte sie sich mit zwei gewaltigen Explosionen selbst.

Der Erstflug der europäischen Trägerrakete Ariane 5 endete am 4. Juni 1996 in einem Desaster. Was war passiert? Die Software hatte sich verrechnet. Der Versuch einer Kurskorrektur ließ die Triebwerke bis zum Maximalwert hochfahren, führte dann zu einer Schieflage und ließ schlussendlich die Rakete sich selbst zerstören.

Auch hier können die Gründe für den Absturz auf unsachgemäßen Umgang mit einem alten System zurückgeführt werden. So wurde die von der Ariane 4 übernommene Software in verschiedenen Belangen nicht korrekt für den Einsatz für die Ariane 5 angepasst und getestet. Im Einzelnen waren dies eine ungeschützte Variable bezüglich der Konvertierung, fehlende Tests mit aktuellen Flugbahndaten sowie ein aktives Unterprogramm zu einem unnötigen Zeitpunkt.

Der Schaden war enorm: Die Entwicklung der Rakete dauerte zehn Jahre und verschlang 5,9 Milliarden Euro. Der erste kommerzielle Flug der Ariane 5 fand dadurch erst drei Jahre später, am 10. Dezember 1999, statt.

Optimisten mögen denken, dass es sich bei den obigen Beispielen um einzelne Ausnahmen handelte. Dass dem nicht so ist, zeigt die enorme Verbreitung solcher großen, älteren Systeme in allen Bereichen unserer Gesellschaft und damit das potenzielle Risiko weiterer solcher Pannen. Sie haben sich inzwischen, zum Teil mehr, zum Teil weniger heimlich, in unser Leben eingeschlichen. Denken Sie nur an die großen Systeme der Telefongesellschaften, die Prepaid-Konten unserer Mobiltelefone verwalten. Auch hier könnten Fehler zu großen Problemen führen. Fallen solche Systeme aus, können Hunderttausende von Abonnenten nicht mehr telefonieren oder schlimmer noch, die Abrechnungen sind fehlerhaft. Ein anderes Beispiel sind die Systeme der Aktienbörsen, die tagtäglich riesige Mengen von Transaktionen verarbeiten. Hierbei wechseln Sekunde für Sekunde sehr viele Wertpapiere den Besitzer. Auch die Verkehrstechnik ist tagtäglich auf das einwandfreie Funktionieren ihrer Applikationen angewiesen. So steuern und überwachen bei den städtischen Verkehrsbetrieben große Leitsysteme den ganzen Bus- und Straßenbahnverkehr. Zusätzlich unterstützen diese Systeme den Funkverkehr zu den Busfahrern sowie Fahrgästen. Wer kennt schließlich nicht die nette Stimme der Züricher Trams, die sagt: »Eine Durchsage der Leitstelle: …« Aber auch die Eisenbahn kann einen reibungslosen Verkehr nur dank intakter Programme gewährleisten. Schließlich passieren schnell über tausend Züge pro Tag einen einzelnen Bahnhof. Weitere solcher Systeme findet man in der Energietechnik bei den Atomkraftwerken, in der Flugindustrie im Cockpit, in großen Liftsteuerungen, auf Hochseeschiffen, in der Autoindustrie in Geschwindigkeitsreglern und Motorsteuerungen bis hin zur Medizintechnik mit ihrer Röntgentechnologie und Computertomografen oder der Software für Hörgeräte und Herzschrittmacher.

Diese Liste ließe sich beliebig erweitern. Sie soll nur zeigen, wie weitverbreitet große Softwaresysteme sind und welche zentralen Aufgaben sie in unserer Gesellschaft bereits wahrnehmen. Mit dieser enormen Verbreitung steigt entsprechend

die Wahrscheinlichkeit von Pannen. Natürlich müssen diese nicht immer so verheerende Folgen haben wie explodierende Weltraumraketen, abstürzende Flugzeuge, in voller Fahrt ausgelöste Airbags oder tödlich verstrahlte Patienten. Aber auch der Blackout ganzer Städte, stehende Züge oder nicht funktionierende Telefonanlagen sind höchst unerwünschte Ereignisse, die oft enorme Kosten und einen entsprechenden Imageschaden mit sich bringen.

Entsprechend wichtig ist es, dass die Softwareindustrie, also wir, in der Lage ist, mit solchen Systemen richtig umzugehen.

1.3 Ursachen für Altersschwäche

Um zu verstehen, wie man mit alternden Systemen besser umgehen und mögliche Gebrechen reduzieren kann, ist es sinnvoll, dass wir verstehen, welche Ursachen zugrunde liegen können.

Was haben unsere Beispiele und ganz generell solche älteren Systeme gemeinsam? Alle handeln von großen, bestehenden Applikationen, die erweitert wurden. Die Komplexität der Applikation ist häufig schon durch die reine Größe der Systeme gegeben. Erschwerend kommt hinzu, dass die ursprünglichen Entwickler gar nicht mehr oder nur noch zum Teil im Team vorhanden sind. Somit fehlt es an Wissen. Aus Kosten- und Zeitgründen hat man auf eine durchgängige Dokumentation verzichtet oder sie ist hoffnungslos veraltet, was den Know-how-Transfer nicht gerade vereinfacht. Viele Fehler wären auch vermeidbar, würden die Entwickler sorgfältiger Sicherheitsabfragen in ihre Programme einbauen (z. B. Design by Contract). Divisionen durch null, Speicherüberlauf durch zu lange Zahlen oder falsch umgewandelte Werte sind typische Beispiele dafür. Oft bleiben diese lange unentdeckt und schlagen ganz gemäß Murphy im ungünstigsten Moment zu. Aber auch das Wiederverwenden von alten Codeteilen, sogenanntes Coderecycling, in neuen Systemen hat seine Tücken. Nicht selten stimmen die ursprünglichen Rahmenbedingungen des kopierten Codes nicht mehr mit dem neuen System überein, was isoliert an diesem Codefragment gar nicht erkannt wird. Auch fehlende oder ungenügende Tests führen oft zu vermeidbaren Computerpannen. Ebenso häufig trifft man schlecht abgesprochene Schnittstellen an. Dies muss nicht nur innerhalb der eigenen Applikation, sondern kann auch zu Fremdsystemen sein. Gewisse Technologien sind eventuell am Ende des Lebenszyklus angelangt und werden von der Lieferfirma nicht mehr unterstützt, oder es passt schlicht die Software nicht zur Hardware. Das Gleiche gilt für alte Programme, die auf einer neuen Version des Betriebssystems eingesetzt werden. Eine weitere Problematik stellt das unzureichend beherrschte Patchmanagement der Betriebssysteme dar. Nicht zu vergessen ist der in den vergangenen Jahren stetig gestiegene Zeit- und Kostendruck, der ebenfalls seinen Beitrag zu dieser unheilvollen Entwicklung beisteuert. Oft ist es aber auch das fehlende Wissen der Infor-

matiker, welche Risiken solche Systeme beinhalten könnten und wie man diesen Risiken begegnen kann.

Wie wir gesehen haben, gibt es vielerlei Gründe für die Altersschwäche unserer Software. Auf der einen Seite betreffen die Herausforderungen Managementthemen wie der Umgang mit Wissen und Mitarbeitern, Organisation, Prozessen und Kosten. Auf der anderen Seite sind es technische Themen wie Codeanalyse, Refactoring, Reengineering, Wartungsmetriken, aber auch Tests aus Wartungssicht. Auf all diese Themen wollen wir in diesem Buch eingehen.

1.4 Wegweiser durch dieses Buch

Wir haben gesehen, dass erst die rasante Entwicklung der Informationstechnologie der letzten Jahre und Jahrzehnte die Möglichkeiten mit sich gebracht hat, komplexe Prozesse mittels Software zu lösen. Gleichzeitig mit diesen neuen Möglichkeiten fanden solche großen Applikationen immer neue Anwendungsgebiete und somit eine enorme Verbreitung – diese Systeme sind heute inhärenter Teil unserer Gesellschaft. Umso empfindlicher trifft es uns alle, wenn diese Anwendungen sich fehlerhaft verhalten oder gar ausfallen. Sie sind also die Stütze unserer modernen Gesellschaft und gleichzeitig unsere Achillesferse. Daher ist die folgende Fragestellung von zentraler Bedeutung: Wie können wir verhindern, dass alternde Softwaresysteme unnötige Probleme und Kosten verursachen?

Seien wir realistisch: Alternde Systeme wird es immer geben. Wir werden sie nicht verhindern können, sondern müssen lernen, mit ihnen zu leben, umzugehen und sie zu pflegen. Wir müssen uns sozusagen mit »Softwaregeriatrie« beschäftigen.Das führt uns zum Thema dieses Buches:

»**Softwarewartung**«

Mit diesem Buch möchten wir eine umfängliche Übersicht über die Softwarewartung geben. Es soll quasi als geistige Landkarte zum Thema der Softwarewartung dienen und die Zusammenhänge der einzelnen Themen zueinander herstellen. Einzelne spezialisierte Themen werden zwar erläutert und in Beziehung zu anderen gesetzt, es wird dazu aber auf weiterführende Literatur verwiesen.

In diesem Buch sprechen wir sowohl technische wie auch kommerzielle Systeme an. Da aus unserer Sicht die Unterschiede zwischen diesen beiden Welten bzgl. der Softwarewartung marginal sind, differenzieren wir nicht weiter. Daher finden Sie an einigen Stellen Begriffe, die eher aus dem Umfeld der technischen Software stammen, wie beispielsweise Inbetriebnahme oder Einsatz im Feld. Dies würde man im Umfeld der kommerziellen Systeme eher Produktivbetrieb nennen. Wir gehen aber davon aus, dass Sie als Leser in der Lage sind, solche Begriffe für Ihr gewohntes Umfeld zu adaptieren.

Zudem betrachten wir in diesem Buch eher große Softwaresysteme mit einem langen Lebenszyklus, die auch gleichzeitig in unterschiedlichen Versionen im Feld vorkommen können. Das heißt nicht, dass dieses Buch für kleinere Systeme uninteressant wäre – die möglichen Schwierigkeiten und Lösungsansätze lassen sich aber an großen Systemen besser zeigen.

Die Auswirkungen der aktuellen agilen Methoden auf die Alterung und Wartung von Softwaresystemen sind noch zu wenig bekannt und können deshalb nur vermutet werden. Wir möchten ausdrücklich festhalten, dass wir hier nicht zwischen den unterschiedlichen Vorgehensweisen werten wollen. Wie sich der agile Ansatz auf die Softwarewartung letztendlich auswirken wird, werden wir erst in fünf bis zehn Jahren wissen.

Zum Inhalt

An wen richtet sich das Buch? An alle, die amateurhafte, undisziplinierte und ineffiziente Wartung nicht ertragen können. Diese Attribute möchten wir anhand der Aufteilung in Abbildung 1–3 kurz erläutern:

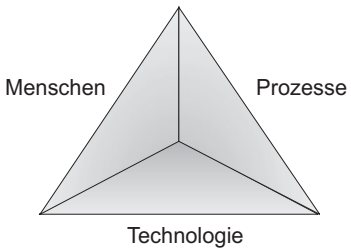

Abb. 1–3 *Grundbausteine der Softwarewartung*

Um Softwarewartung produktiv betreiben zu können, muss allen drei Erfolgsfaktoren ausreichend Rechnung getragen werden. Diese drei Erfolgsfaktoren sind: Menschen, Prozesse und Technologie:

- Um Softwarewartung erfolgreich betreiben zu können, braucht es gut ausgebildete Mitarbeiter. Sind diese nicht vorhanden, stehen einem offensichtlich *Amateure* zur Verfügung. Somit ist einer der Erfolgsfaktoren nicht erfüllt.
- Genauso verhält es sich mit den Prozessen. Diese müssen klar definiert sein und gelebt werden. Nur so ist ein produktives Arbeiten möglich. Ist diese Voraussetzung nicht erfüllt, betreibt man eine *undisziplinierte* Wartung.
- Der dritte Faktor betrifft die Technologie. Hier sind angepasste, nützliche und wirtschaftliche Werkzeuge gefragt. Sind diese nicht vorhanden, betreibt man eine *ineffiziente* Wartung.

Eine professionelle, disziplinierte und effiziente Softwarewartung ist also nur möglich, wenn alle drei Faktoren genügend berücksichtigt werden. Daher möchten wir

diese ganzheitliche Sicht mit möglichst vielen Facetten in diesem Buch darstellen. Entsprechend haben wir das Buch in drei Teile gegliedert (s. Abb. 1–4).

- Teil I – Grundlagen der Softwarewartung
- Teil II – Managementthemen der Softwarewartung
- Teil III – Techniken der Softwarewartung

Alle drei Teile sind gleich wichtig und bestehen jeweils aus vier Kapiteln. Diese Abbildung ist einerseits eine Orientierungshilfe zum Buch und andererseits spiegelt sie unser Verständnis der Zusammenhänge der verschiedenen Subthemen der Softwarewartung wider.

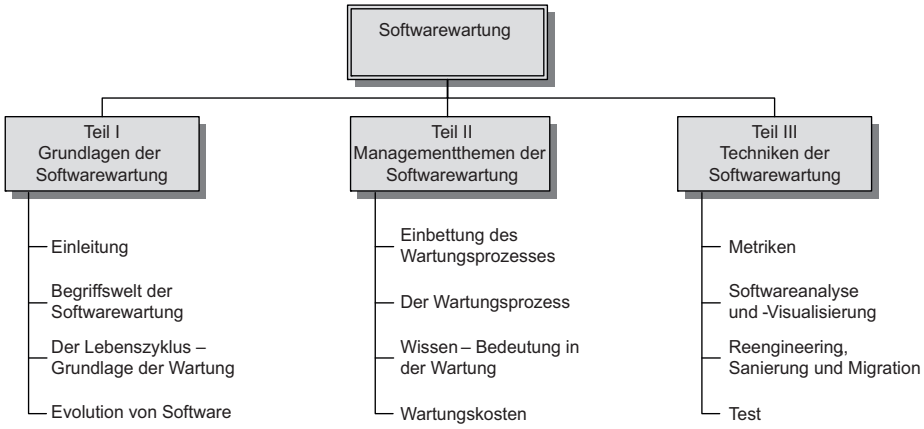

Abb. 1–4 *Überblick über das Buch*

Der erste Teil erklärt zunächst die Gründe, die eine Wartung nötig machen, sowie die wichtigsten Begriffe der Softwarewartung. Anschließend wird auf die Wichtigkeit des Lebenszyklus eingegangen, der die absolute Grundlage für das vertiefte Verständnis der Softwarewartung darstellt. Zum Schluss schauen wir uns die Grundsätze der Evolutionslehre an. Alles in allem gehen wir in diesem Abschnitt auf die grundlegenden Themen und Begriffe rund um die Softwarewartung ein.

Teil II befasst sich mit den Managementthemen rund um die Softwarewartung. Hier sprechen wir Themen wie den Wartungsprozess, mögliche Wartungsorganisationen und den Umgang mit den Wartungskosten an, aber auch weichere Faktoren wie die Bedeutung des Wissens im Umfeld der Wartung, den Umgang mit Gurus oder auch die Mitarbeitermotivation.

Im dritten Teil geht es mehr um die verschiedenen Wartungstechniken. Wir zeigen Monitoring-Möglichkeiten mittels Metriken auf, besprechen Analyse- und Visualisierungstechniken, gehen auf die Bedeutung des Reengineering, der Sanierung sowie der Migration im Rahmen der Softwarewartung ein und beleuchten

zum Schluss Aspekte des Testens in demselben Kontext. Viele dieser Techniken werden kurz erklärt und in Beziehung gesetzt. Für eine Vertiefung dieser Themen wird jeweils auf die entsprechende, spezialisierte Literatur verwiesen.

Diese Einteilung erlaubt es dem technisch orientierten Leser beispielsweise, nur die Teile I und III zu lesen oder dem am Management interessierten Leser die Teile I und II. Natürlich freut es uns besonders, wenn man das Buch einfach durchliest.

Lesehilfe

Zur Unterstützung für den Leser bieten wir in diesem Buch zwei Elemente an. Zum einen heben wir an verschiedenen Stellen die aus unserer Sicht wichtigsten Aussagen der einzelnen Kapitel mittels eines grauen Kastens grafisch hervor:

Hier steht ein Merksatz

Zum anderen finden Sie am Schluss jedes Kapitels eine knappe Zusammenfassung mit den jeweils wichtigsten Aussagen.

Nun wünschen wir Ihnen viel Spaß mit diesem Buch!