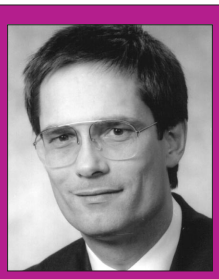
# Muß Software wirklich Fehler haben?

## Auseinandersetzung mit einem Glaubensgrundsatz der Softwareindustrie



Martin Rösch

**Software kann nicht fehlerfrei sein! Warum eigentlich nicht? Dieser Artikel rüttelt an einer fest gefügten Überzeugung vieler Softwareentwickler. Er soll in konstruktiver Weise zeigen, daß fehlerfreie Software ein seriöses Ziel der Softwareentwicklung sein kann und beschreibt einen Weg zu diesem Ziel.**

Würden Sie mit einem Flugzeug fliegen, das so gebaut ist, wie die Software von heute? Diese Frage werden einige von uns vielleicht schon im Jahr 2002 mit einem bedenkenlosen JA beantworten, wenn Objektorientierung und Qualität ihre selbstverständlichen Plätze in der Welt der Softwareentwicklung eingenommen haben. Sowohl Objektorientierung als auch Qualitätsdenken sind notwendige Voraussetzungen, keines von beiden darf fehlen: Objektorientierung gibt die technischen Grundlagen, und die Spielregeln (siehe Kasten) stammen aus dem Qualitätsdenken.

Diese Spielregeln zeigen schon recht deutlich, was uns bei der Softwareentwicklung heute fehlt: Wir können meistens nicht einmal sagen, ob eine Software ihre Anforderungen erfüllt oder nicht, denn die Anforderungen sind meist nicht genau und zweifelsfrei bekannt. Und selbst wenn die Anforderungen bekannt sind, setzt spätestens bei der Abnahme die große Diskussion ein, was denn wohl mit dieser oder jener Anforderung gemeint gewesen sei.

Mercedes Benz hat die A-Klasse vom Markt genommen, weil sie ihre Anforderungen nicht erfüllte: Sie hat den „Elch-Test“ nicht überstanden. Wenn die Softwareentwickler heute ebensoviel Verantwortung für ihre Kunden zeigten, gäbe es in der Softwareindustrie* eine lange Produktionspause. In diesem Licht gesehen, kann unsere Software eher mit den qualmenden Autos von Laurel & Hardy (Dick & Doof) verglichen werden, und wir können schon stolz sein, wenn sie einmal so gut sein wird wie die A-Klasse heute – für die A-Klasse gibt es immerhin überprüfbare Anforderungen.

Ob die Anforderungen eines Softwaresystems das gesamte System beschreiben, ist heute eher vom Zufall abhängig, da Softwareentwickler gerne ergänzen, was ihnen wichtig erscheint. Das war in der Vergangenheit häufig sogar erwünscht, denn das Mitdenken der Softwareentwickler hat ja bekanntlich schon manches Projekt gerettet. Warum sollten wir also mit dieser schönen Tradition brechen? Der Grund liegt darin, daß auf diesem traditionsreichen Weg sehr leicht und schnell ein undokumentiertes Verhalten von Software entsteht. Aber was ist daran denn so schlimm? Undokumentiertes Verhalten heißt, daß das Verhalten von Software nicht vollständig spezifiziert ist, daß sich niemand auf dieses Verhalten verlassen darf, und daß jeder, der es dennoch nutzt, Gefahr läuft, den Boden unter den Füßen zu verlieren.

Undokumentiertes Verhalten entsteht heute meist durch Nachlässigkeit, doch manchmal ist es auch ein unfairer Versuch von Herstellern, ihren Markt gegen die Konkurrenz abzuschotten. Früher wurde das von IBM so praktiziert, und vor kurzem ist Microsoft dabei erwischt worden, daß Word und Excel undokumentierte Windows-Calls benutzen.

Wenn wir hohe Softwarequalität anstreben, darf Software kein undokumentiertes Verhalten aufweisen, und die Benutzer einer Software sollten nur das dokumentierte Verhalten verwenden. Aber wie beschreibt man das Verhalten von Software?

Für die Softwareindustrie (Hersteller und Entwicklungsabteilungen) beginnt das Problem schon damit, daß die Anforderungen für ihre Produkte oft nicht einmal schriftlich fixiert sind. Damit bleibt es der Kreativität und Phantasie einzelner Entwickler überlassen, was eine Software wirklich kann. Die Anforderungen der Anwender bleiben häufig allein schon deshalb unberücksichtigt, weil sich niemand die Mühe macht, sie aufzuschreiben.

Doch auch in den wenigen Unternehmen, in denen jede Softwareentwicklung mit schriftlich fixierten Anforderungen beginnt, ist die Gefahr noch nicht gebannt: Wir haben noch kein Unternehmen kennengelernt, in dem jede fachliche Anforderung wirklich eindeutig mit Hilfe von Akzeptanzkriterien präzisiert wird. Auch in den fortschrittlichsten Unternehmen (nach meiner Einschätzung

### Spielregeln für Qualität

- Ein System oder Objekt muß seine Anforderungen erfüllen.
- Das gesamte Verhalten eines Systems oder Objekts muß durch Anforderungen beschrieben sein.
- Die Anforderungen müssen schriftlich gestellt sein und einen Verantwortlichen haben.
- Zu jeder Anforderung gibt es Akzeptanzkriterien.
- Eine Anforderung ohne Akzeptanzkriterien ist keine.
- Die Akzeptanzkriterien prüfen, ob ihre Anforderung erfüllt ist.
- Eine Anforderung ist erfüllt, wenn alle ihre Akzeptanzkriterien erfüllt sind.

**Dipl.-Inform. Martin Rösch ist Geschäftsführer der „Rösch Consulting Gesellschaft für innovative Softwareentwicklung mbH“ in Deutschland sowie der „Objects 9000 Inc.“, der „Object Academy Inc.“ und der „Rosch Consulting Inc.“ in den USA. Seine E-Mail-Adresse lautet: mr@roesch.com.**

* „Softwareindustrie“ umfaßt alle Softwareentwickler, bei Herstellern ebenso wie in den Softwareabteilungen von Anwenderunternehmen.“

schweizerische Banken) ist die Präzisierung von Anforderungen nur in einzelnen Projekten, aber noch nicht unternehmensweit die Regel. Und auch in den wenigen Projekten, wo bereits auf Qualität (und nicht nur auf Qualitätsmanagement) geachtet wird, beschränkt sich die Qualität auf die fachlichen Anforderungen. Die nicht-funktionalen Softwareanforderungen (für Design und Implementierung) werden bisher in keinem mir bekannten Unternehmen mit der von der ISO 9000 geforderten Präzision aufgenommen – obwohl auch das möglich wäre.

Statt dessen konzentrieren sich viel zu viele Unternehmen auf das Erlangen eines ISO-9000-Zertifikates. Daß dieses Zertifikat nur die Prozesse absichert, mit denen Software erstellt wird, aber keinerlei Aussage über die Qualität der Software selber machen kann, wird allzu oft und allzu geflissentlich übersehen.

## Qualität oder Qualitätsmanagement?

- Qualität braucht Meßbarkeit durch Anforderungen, Akzeptanzkriterien und den Nachweis ihrer Erfüllung.
- Qualität entsteht durch das Lösen von Problemen,
- Qualitätsmanagement stellt sicher, daß die Probleme auch gelöst bleiben und nicht immer wiederkehren.
- Ohne meßbare Qualität ist Qualitätsmanagement eine Illusion.

Der Unterschied zwischen Qualität und Qualitätsmanagement wird in der Softwarewelt noch nicht verstanden. Deshalb sind heute die meisten ISO-9000-Zertifikate für Softwareabteilungen nach meiner Einschätzung wenig wertvoll: Die Zertifikate haben die stolzen Besitzer nur davon abgelenkt, daß die betreffende Entwicklungsabteilung gar keine Qualität produzieren kann, weil sie nämlich entweder keine vollständigen Anforderungen für ihre Software besitzt oder aber die Einhaltung der Anforderungen aus Mangel an Akzeptanzkriterien nicht überprüfen kann. Und die Mitarbeiter verschwenden ihre Zeit mit dem Beschreiben, Lernen und Kontrollieren von Prozessen, von denen jeder weiß, daß sie doch nicht das gewünschte Ergebnis bringen können: richtige Qualität. Hier muß der Hebel angesetzt werden, hier liegt der Schlüssel für echte Qualität: Wer nicht weiß, was er will, muß sich nicht wundern, was er kriegt – so einfach ist das und wird doch so oft vergessen. Anforderungen und Akzeptanzkriterien bilden den Kern der Qualität. Wer nicht beides hat, braucht über Qualitätsmanagement nicht nachzudenken; das Ergebnis wäre bestenfalls eine Illusion, eine Fata Morgana, die Qualität vorspiegelt, wo keine sein kann, weil die Grundlagen fehlen: Anforderungen und Akzeptanzkriterien.

Die Aufgabe der Akzeptanzkriterien einer Anforderung besteht darin, die Erfüllung der Anforderung meßbar zu machen. So, wie Längen in Zentimetern angegeben und mit einem Lineal gemessen werden, und so, wie Gewicht in Kilogramm angegeben und mit einer Waage gemessen wird, so wird Qualität durch Anforderungen beschrieben und mit Hilfe von Akzeptanzkriterien gemessen.

Ein Beispiel: Fünf Zentimeter, gemessen mit einem Lineal, mit einer Schiebelehre oder mit einer Mikrometerschraube sind nicht dasselbe – das weiß jeder Ingenieur. In technischen Zeichnungen wird die Genauigkeit, mit der ein Maß einzuhalten ist, sogar durch eigene Symbole beschrieben.

Für Softwareentwickler wird Präzision dagegen oft zum Ratespiel. Ein Beispiel für eine Anforderung: Eine rund um die Uhr geöffnete Videothek (z. B. in einer Tankstelle) stellt für das Ausleihen eines Videos für jeweils einen ganzen Tag 10 DM in Rechnung. Doch was ist ein ganzer Tag? Erst die Akzeptanzkriterien bringen Klarheit darüber, ob eine Ausleihfrist vom 1. Februar 0:30 Uhr bis zum 3. Februar 23:40 Uhr als ein Tag, als zwei Tage oder gar als drei Tage zu berechnen ist. Ohne Akzeptanzkriterien kann die Entscheidung nicht zweifelsfrei getroffen werden.

Doch Akzeptanzkriterien beschreiben nicht nur präzise, was mit einer Anforderung tatsächlich gemeint ist; sie dienen auch der Überprüfung, ob ein Ergebnis die gestellten Anforderungen erfüllt. Damit ist der neuralgische Punkt der heutigen Softwarequalität – die Erstellung der Testdaten – vom Tisch: Alle fachlichen Testdaten, die für die Überprüfung eines Systems erforderlich sind, sind von Anfang an bekannt, bevor die ersten Entwicklungsarbeiten beginnen.

Hier wenden Kritiker aus der klassischen Softwareentwicklung ein, daß die gegenseitige Beeinflussung verschiedener Anforderungen dazu führt, daß die Überprüfung einzelner Anforderungen – ohne die Berücksichtigung aller Kombinationen mit anderen Anforderungen – Löcher in der Prüfbarkeit erzeugen kann. Als Gegenargument möchte ich das Beispiel einer dicken, schweren Ankerkette aus dem Schiffsbau heranziehen: Hier reicht es ganz offensichtlich aus, wenn wir je zwei benachbarte Ringe auf ihre Festigkeit überprüfen. Der Test wird nicht dadurch besser, daß wir alle möglichen Teilketten der Ankerkette zusätzlich überprüfen. Wenn es Wechselwirkungen zwischen zwei Anforderungen gibt, dann muß es hierzu eine weitere Anforderung geben. Dieses Verfahren funktioniert und hat sich in Projekten bewährt. Kombinationsfehler sind bisher nicht aufgetreten – und ich glaube auch, daß sie bei einer sauberen objektorientierten Bauweise nicht auftreten werden. Wichtig für diese Vereinfachung der Testmethodik ist aber die objektorientierte Bauweise von Software und die hierdurch erreichte konsequente Kapselung, die auch weiter unten noch einmal eine Rolle spielen wird. Mit der herkömmlichen Art der Softwareerstellung wäre eine solche Vereinfachung der Testmethodik nicht zu verantworten.

Wenn alle Akzeptanzkriterien einer Anforderung erfolgreich geprüft wurden, gilt die Anforderung nach ISO 9000 als erfüllt. Hier liegt eine Verantwortung der Anforderungssteller, die heute meist noch nicht seriös wahrgenommen wird: Nach ISO 9000 ist der Auftraggeber für die Anforderungen und für die Vollständigkeit der Akzeptanzkriterien zu einer Anforderung verantwortlich. Dennoch entziehen sich noch immer viele Anwender dieser Verantwortung, und die Softwareentwickler laufen bereitwillig in die aufgebaute Falle: Woher soll ein Softwareentwickler denn wissen, wie z. B. das Risikomanagement bei bestimmten Kreditgeschäften funktioniert? Wer sich hier als Entwickler mit einem „macht ihr das mal“ den Affen auf die Schulter setzen läßt, geht Risiken für das Unternehmen ein, die er nicht abschätzen und auch nicht tragen kann. Auf der anderen Seite spielt eine Fachabteilung, die nicht sagen kann, wie ihr Geschäft genau funktioniert, mit ihrer Daseinsberechtigung. Das fachliche Geschäft ist heute vielfach schon derart schnell und komplex, daß nur die kompetenten Mitarbeiter in den Fachabteilungen wirklich sagen können, was richtig und was wichtig ist. Softwareentwickler sollten nur noch dafür sorgen, daß diese Aussagen schon vor der Systementwicklung auf dem Tisch liegen und nicht erst bei der Abnahme zu Überraschungen führen.

Leider kommt es in der Praxis häufig vor, daß Anforderungen zwar gestellt, aber nicht durch Akzeptanzkriterien präzisiert werden können. Entwickler täten gut daran, solche Anforderungen zurückzuweisen. Wenn ein Auftraggeber nicht vorher weiß, wie die Er-

füllung einer Anforderung überprüft werden kann, wie weiß er es dann nachher bei der Abnahme? Entweder testet er die Anforderung nicht (dann hätte er sie auch weglassen können) oder es gibt nachher doch Prüfbedingungen – die hätten aber vorher auf den Tisch gehört. Sie nachher aus der Schublade zu ziehen, ist nach der ISO 9000 nicht möglich.

## Technische Grundlagen aus der Objektorientierung

- Objekte verstecken technische Details.
- Objekte werden erst gedacht, dann gebaut.
- Objekte repräsentieren menschliche Denkstrukturen.
- Objekte können unter Benutzung anderer Objekte erstellt werden.
- Die Eigenschaften von Objekten sind vollständig beschreibbar.
- Die Komplexität jeder Montagesituation kann begrenzt werden.
- Standards wie CORBA oder DCOM beschreiben die Schnittstellen.

Zusammen mit den Spielregeln aus dem Qualitätsdenken ermöglichen diese technischen Grundlagen den Bau von Software, von der Entwickler nachweisen können, daß sie alle gestellten Anforderungen erfüllt.

So, wie bei der Ankerkette die interne molekulare Struktur jedes Rings in ihm selber verborgen bleibt, so halten auch Objekte ihr Inneres geheim. Nach außen wird nur das dokumentierte Verhalten sichtbar. Wenn nun das Verhalten vollständig mit Anforderungen und Akzeptanzkriterien beschrieben und überprüft ist, dann kann man sich auf das Objekt (genauer: seine Klassendefinition) verlassen. Vielleicht muß jeder einzelne Ring der Ankerkette mit Ultraschall oder Röntgen auf seine interne Beschaffenheit geprüft werden, aber, wenn er einmal geprüft ist, dann weiß man, was man hat: einen Ring einer Ankerkette mit bekannten Eigenschaften. Wenn ich mehrere solcher geprüften Ringe zu einer Kette zusammenfüge, kann ich die molekulare Struktur jedes einzelnen Rings außer acht lassen: Sie ist weggekapselt, und nach außen interessiert nur noch die Einhaltung der Anforderungen, wie z. B. eine bestimmte Reißfestigkeit.

Die Objektorientierung gibt uns die Möglichkeit, das Innere eines Objekts so zu verbergen, daß keine undokumentierten Wechselwirkungen zwischen dem Inneren des Objekts und seiner Umwelt auftreten können. Durch diese Fähigkeit der Objektorientierung werden wir in die Lage versetzt, mit bekannten und geprüften Bausteinen arbeiten zu können – eine unabdingbare Voraussetzung für qualitativ hochwertiges Arbeiten und für nachweisbar fehlerfreie Ergebnisse.

Die schönsten Objekte helfen uns aber wenig, wenn sie technische Kunstgebilde sind, die keiner mehr so ganz versteht. Deshalb hat es sich bewährt, Objekte immer nur nach den Denkstrukturen von Menschen zu modellieren. Das gilt sowohl für fachliche Objekte, wie z. B. die Business-Objekte und ihre Helfer, als auch für technische Objekte, die erst bei der Implementierung benutzt werden und deren Einsatz im Design geplant wird. Am Anfang eines brauchbaren Objekts steht also immer ein Mensch mit seinem Denken. Andere Objekte müssen scheitern, weil sie beim Verstehen stören, und weil sie künstlich kompliziert sind.

Die Softwareentwicklung muß wieder mehr auf die beteiligten Menschen Rücksicht nehmen. Menschen können nun einmal nur eine begrenzte Komplexität bewältigen. In ihrem Denken können sie bis zu ca. sieben Faktoren gleichzeitig berücksichtigen. $7 \pm 2$ ist schon 1956 als Grenze der menschlichen Denkfähigkeit dokumentiert worden. Danach beginnen die Fehler – wie bisher bei der Softwareentwicklung üblich (vgl. [Mil56]).

In anderen Branchen ist die Komponentenbildung schon lange üblich, und man hat auch die Qualitätsfrage schon lange im Griff (vgl. [Wom90]): Ein Auto besteht heute aus ca. 40.000 Teilen, doch in der Endmontage werden nur noch 800 Teile zusammengesetzt. Deshalb ist der heutige Trend zu Komponentensoftware auch richtig und notwendig. Nur sollte dabei nicht vergessen werden, daß Komponenten nur dann sinnvoll sind, wenn sie eine definierte Qualität aufweisen – und da liegt heute noch der Hase im Pfeffer. Fragen Sie Ihre Komponentenanbieter doch einmal nach vollständigen Spezifikationen in Form von Anforderungen, Akzeptanzkriterien und dem Nachweis der Verifikation (siehe Kasten).
Diese Fragen gelten für interne Lieferanten ebenso wie für externe, d. h. für IBM und Microsoft und SAP ebenso wie für die hausinternen Entwickler von Software.

## Fragen an Komponentenanbieter

- Welche Anforderungen erfüllt Ihre Komponente?
- Mit welchen Akzeptanzkriterien messen Sie die Erfüllung der Anforderungen?
- Weisen Sie uns bitte nach, daß die Komponente bei Ihnen im Haus die Anforderungen erfüllt hat.
- Können Sie garantieren, daß die Komponente kein undokumentiertes Verhalten aufweist?
- Von welchen Einsatzbedingungen ist das korrekte Funktionieren Ihrer Komponente abhängig?
- Wie können wir sicherstellen, daß unser Umfeld die Einsatzbedingungen Ihrer Komponente erfüllt?
- Wie können wir überprüfen, ob die Komponente auch in unserem Umfeld alle Anforderungen erfüllt?

Die Objektorientierung gibt uns jetzt die Chance, auch den Zusammenbau von Software – wie den Automobilbau – in überschaubare Einheiten zu zerlegen, von denen jede mit hoher Prozeßsicherheit und nachweisbarer Qualität erstellt werden kann. Das gilt sowohl für die „Objektorientierung-im-Kleinen“, die vor ca. 30 Jahren (vgl. [Dah67] und [simula]) mit den Programmiersprachen begann, als auch für die „Objektorientierung-im-Großen“ der Business-Objekte, die seit 1992 möglich ist (vgl. [Har93]).

Durch Anwendung des Kapselungsprinzips war es bisher schon möglich, die Korrektheit von objektorientierten Analysemodellen nachzuweisen (vgl. [objects]). Erste Überlegungen und laufende Versuche legen die Vermutung nahe, daß die Erfahrungen aus der objektorientierten Analyse bei geeigneten Randbedingungen auch auf das objektorientierte Design und die objektorientierte Implementierung übertragen werden können.

Hierfür ist es notwendig, die einzelnen Arbeitsschritte so klein und überschaubar zu gestalten, daß die Bedingungen für ein fehlerfreies Arbeiten noch beschreibbar sind. Erfahrungen aus anderen Industrien zeigen, daß dies möglich ist. Es ist zwar nicht trivial, aber auch nicht so unmöglich wie in der bisherigen Softwareentwicklung. Wenn dies gelingt, wird die Komplexität jeder einzelnen Montagesituation so weit begrenzt wer-

den, daß eine vollständige und genaue Beschreibung und damit auch die Überprüfbarkeit in den Bereich des Möglichen gelangt.

Unterstützend wirken hierfür Standards, wie z. B. CORBA oder auch DCOM, die dabei helfen, das Innere von Softwarekapseln zu verpacken. Dabei ist es relativ egal, ob CORBA oder DCOM als Verpackung gewählt wird. Wichtig ist nur, daß die gekapselten Objekte einem menschlichen Denkvorgang entspringen und keine technisch motivierten Kunstkonstrukte sind. Dann können, wie bereits diverse Produkte gezeigt haben, CORBA- und DCOM-Objekte sehr leicht umgepackt werden – sie bleiben ja dieselben Objekte. Deshalb ist die gegenwärtige Auseinandersetzung „CORBA vs. DCOM" genauso unwichtig wie der früher so gerne geführte Streit um OOA-Notationen.

## Kritik am bisherigen „Beweisen von Software"

- Die Anforderungen kommen nicht von den Anwendern.
- Meßlatte ist ein logischer Ausdruck, der von Anwendern sowieso nicht mehr verstanden wird.
- Die Möglichkeiten der Objektorientierung werden nicht berücksichtigt.
- Unrealistische Qualitätsmaßstäbe verstellen den Blick auf das Machbare.
- Auch bei Flugzeugen, Autos, Häusern und Aufzügen bringen die Software-Beweisverfahren negative Ergebnisse.
- Warum sollen wir den Verfahren dann bei Software vertrauen?

Die bisherigen Versuche zum Beweisen von Software können als gescheitert bzw. praxisuntauglich angesehen werden. Doch der Umkehrschluß, daß der Nachweis von Korrektheit für Software unmöglich sei, ist meiner Ansicht nach falsch – auch wenn er im Denken der gegenwärtigen Informatikergeneration tief verwurzelt zu sein scheint. Oder ist diese Schlußfolgerung eine willkommene Ausrede, um uns trotz der Unzulänglichkeiten unserer Arbeit den Glorienschein „Programmierer – Elite der Automation" (vgl. [Ste70]) umzuhängen?

Die Berichte über das Beweisen von Software haben doch nur gezeigt, daß es nicht möglich war, die in den Berichten geforderten Nachweise zu führen. Vielleicht waren ja auch nur die Nachweisforderungen falsch. Versuchen Sie doch einmal, zu beweisen, daß ein Auto niemals in einen Unfall verwickelt sein wird, oder daß es niemals von der Straße abkommen wird! Wir alle wissen, daß das nicht geht. Können wir deshalb keine Autos bauen? Doch, können wir; und im Vergleich zu unserer Software sind unsere Autos schon sehr fehlerfrei und außerordentlich zuverlässig. Dasselbe gilt für praktisch alle anderen Industrien – mit Ausnahme der Softwareindustrie.

Ich gehe davon aus, daß es noch vor dem Jahr 2000 möglich sein wird, auch in Softwareprojekten – zumindest in ersten Pilotprojekten – denselben Qualitätsstandard zu erreichen, der für den Rest der Industrie heute schon selbstverständlich ist.

## Zusammenfassung

Die Voraussetzungen für nachweisbar korrekte Softwaresysteme sind geschaffen: Objektorientierung und die ISO 9000 geben uns die technischen Konstruktionsverfahren sowie die Spielregeln für den Umgang mit Qualität. Die konsequente Anwendung dieser beiden Grundlagen wird nach meiner Einschätzung dazu führen, daß der gesamte Software-Konstruktionsprozeß aus überschaubaren, überprüfbaren und nachweisbar korrekt durchgeführten Schritten besteht. Es erscheint möglich, daß schon in naher Zukunft nachweisbar korrekte Software erstellt werden kann und nicht nur nachweisbar korrekte Analysemodelle – die gibt es schon heute.

## Ausblick

Wenn die ersten Projekte mit nachweisbar korrekter Software erfolgreich gelaufen sind, wird die heutig gängige Überzeugung „Software kann nicht fehlerfrei sein" nicht mehr haltbar sein. Sie wird dann meiner Meinung nach als überzogene Anforderung erkannt, die eine ganze Branche 20 Jahre lang gelähmt hat.

Mit dem Thema „Qualität und Qualitätsmanagement" setzt Martin Rösch sich auch im Rahmen der OOP '98 in München auseinander. Sein Vortrag, der am 10. Februar 1998 stattfindet, trägt den Titel „Fehlerfreie Software – erste Erfahrungen" (vgl. auch Anzeige auf S. 43-50).

## Literatur

[**Dah67**] O.-J. Dahl, K. Nygaard, Class and subclass declarations, Norwegian Computing Center Document, March 1967

[**Har93**] P. Harmon, Objects In Action: Commercial Applications Of Object-Oriented Technologies, Addison-Wesley, 1993

[**Mil56**] G.A. Miller, The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information, Psychological Rewiew, 1956

[**objects**] Erste Erfahrungen mit der Fehlerfreiheit von Software und andere Qualitäts-Links, siehe: http://www.objects9000.com

[**simula**] Bericht über die Anfänge von Simulua, siehe: http://lingua.utdallas.edu/articles/simula.html#fn62

[**Ste70**] K. Steinbuch, Programmierer – Elite der Automation?, dtv, ca. 1970

[**Wom90**] J.P. Womack et. al., Die zweite Revolution in der Automobilindustrie, Heyne-Taschenbuch, 1990