erschienen in der FIfF-Kommunikation,
herausgegeben von FIfF e.V. - ISSN 0938-3476
www.fiff.de

Hans-Jörg Kreowski

Falsch taugt nichts –
Eine Glosse zum Fortschritt der Softwaretechnik

Ende der 60er, Anfang der 70er Jahre des 20. Jahrhunderts machte in der Informatik das Wort von der *Softwarekrise* die Runde. Gemeint war der Umstand, dass die Entwicklung von Software mit der Entwicklung von Hardware kaum mithalten konnte, vergleichsweise teuer war und im Ergebnis an Brauchbarkeit und Zuverlässigkeit zu wünschen übrig ließ. Aufgefallen ist das vor allem auch im militärischen Bereich, in dem schon frühzeitig große, komplexe Systeme geplant wurden, die nur mit vielen Beteiligten über lange Zeiträume hinweg programmiert werden konnten. Es wurde evident, dass es an Methoden mangelt, mit denen sich große Programmsysteme in großen Teams mit langen Entwicklungszeiten systematisch so herstellen ließen, dass ihre Kosten im Rahmen bleiben und die fertigen Programme die intendierten Aufgaben reibungslos und zuverlässig erledigen. Um eine solche Methodik zu erreichen, wurde das Gebiet der Softwaretechnik aus der Taufe gehoben. Mit ihr kamen auch sehr schnell die ersten Zauberformeln auf, mit denen das Problem gelöst sein sollte, wie die *systematische Programmierung*, die *strukturierte Programmierung*, die *Modularisierung*, *abstrakte Datentypen* u. ä. Ein weiteres Zauberwort war *Korrektheit*, wobei ein Programm korrekt genannt wird, wenn es tut, was es soll, wenn also das Verhalten des Programms den gestellten Anforderungen genügt. Allerdings ging spätestens an dieser Stelle ein tiefer Riss durch die Gemeinde der Softwaretechnikerinnen und -techniker. Eine verschwindend kleine Minderheit theoretisch orientierter Fachleute wies darauf hin, dass Korrektheit nur dann sichergestellt werden kann, wenn man das Programmverhalten und die Anforderung formalisiert und ihre Übereinstimmung im mathematischen Sinne beweist. Die übergroße Mehrheit hatte dafür lediglich Unverständnis, ein mildes Lächeln oder Hohngelächter übrig. Die Debatte um die Softwaretechnik auf den Konferenzen zwischen 1970 und 1980 – zumindest soweit ich sie besucht und in Erinnerung behalten habe – war geprägt vom allgemeinen Gejammer über die Probleme, von der Anhäufung hohler Phrasen, von der massiven Zurückweisung jedes einzelnen konstruktiven Vorschlags zur Verbesserung und dem nahezu einhelligen Einvernehmen, dass Verifikation für große Programme nicht machbar ist.

Und wie steht es heute rund 40 Jahre später um die Softwaretechnik? Auf den ersten Blick hat sich kaum etwas geändert. Die Entwicklung von Software hinkt der Hardware immer noch hinterher. Die Entwicklung von Software ist immer noch teuer und die Kosten laufen häufig aus dem Ruder. Und was die Zuverlässigkeit angeht, bleiben weiterhin viele Wünsche offen. Es gibt diverse Beispiele für Projekte, bei denen Hunderte Millionen Euro verschwendet wurden, weil die entwickelten Systeme nie oder zu spät zum Einsatz kamen. Aber das ist ohnehin nur die Spitze des Eisbergs, denn viele Systeme im alltäglichen Einsatz sind fehleranfällig und verärgern ihre Benutzerinnen und Benutzer durch ungewöhnliches, ungeahntes und unakzeptables Verhalten. Gibt es eigentlich wissenschaftlich fundierte Untersuchungen darüber, wie groß die finanziellen Verluste sind, die durch fehlerhafte, nicht funktionierende oder lediglich bedingt brauchbare Software entstehen? Und Geld ist nicht alles; Programmfehler können auch lebensbedrohlich sein. Gibt es eigentlich wissenschaftlich fundierte Untersuchungen darüber, wie viele Menschen durch Softwarefehler ums Leben kommen?

Aber in den letzten 40 Jahren hat sich auch einiges getan. Die Minderheit der theoretisch orientierten Fachleute ist nicht mehr verschwindend klein, sondern nur noch klein. Über die Idee, Software zu verifizieren, um ihre Zuverlässigkeit zu verbessern, wird nicht mehr so laut gelacht und das Unverständnis ist eingedämmt. Aus dem Kreis der Mehrheit ist heute manchmal schon zu hören, dass es gut wäre, Methoden zu haben, die die Verifikation erlauben. Und wenn man an *Model Checking* und *Theorem Proving* denkt, dann ist inzwischen sogar die Idee der Verifikation in bescheidenem Umfang realisiert und im praktischen Einsatz erfolgreich. Aber die große Mehrheit der Softwareentwicklerinnen und –entwickler baut auch heute noch auf Techniken und Methoden, die nur informell erklärt sind und bei denen unklar ist, ob die dafür verwendeten Werkzeuge auch das durch die Methoden beschriebene Verhalten umsetzen. Selbst wenn die Methoden eine formale Semantik besitzen, wird diese häufig bei der Entwicklung von Softwaresystemen ignoriert. Was für die einzelnen Methoden schon fahrlässig ist, potenziert sich noch beim Zusammenspiel mehrerer Methoden. Ein Methodenmix ist bei der Entwicklung großer Softwaresysteme praktisch unvermeidlich, in seinen Wirkungen aber in den allermeisten Fällen weder erklärt noch gar formal beschrieben. Diese Kritik bezieht sich auf alle Softwareentwicklungen mit UML-Diagrammen, ereignisgesteuerten Prozessketten oder mit sonstigen sogenannten semi-formalen Modellierungskonstrukten. Aber auch der Einsatz von Petri-Netzen, abstrakten Datentypen oder sonstigen formalen Methoden kann gar nicht ausgenommen werden, weil bei deren Verwendung die formale Semantik häufig ausgeblendet wird. Um nicht missverstanden zu werden, sei angemerkt, dass diese Modellierungsmethoden immer noch besser sind als ein planloses Ad-hoc-Programmieren und einen Fortschritt darstellen gegenüber einem methodenlosen Vorgehen. Nicht die Methoden sind schlecht; sondern die Einstellung ihrer Nutzerinnen und Nutzer ist zu kritisieren, die glauben, durch Verwendung dieser Methoden darum herumzukommen, Software zu verifizieren, wenn ihre Leistung sicher sein soll. Ein Programm kann schnell laufen und wenig Speicherplatz verbrauchen, es kann eine benutzungsfreundliche Oberfläche aufweisen, es kann wohlstrukturiert und modular aufgebaut sein, es kann bezüglich aller nur erdenklichen Qualitätsmerkmale äußerst positiv abschneiden; es taugt dennoch nicht, wenn es falsch ist. Anders gesagt, wer nicht das Hauptaugenmerk bei der Systementwicklung auf die Korrektheit der entwickelten Systeme legt, nimmt fehlerhafte Programme billigend und fahrlässig in Kauf. Qualitätssicherung ohne Korrektheit bleibt eine Farce. Die große Mehrheit in der Softwaretechnik bleibt auf dem Holzweg, solange nicht die Herausforderung der Systemverifikation angenommen wird und verifizierte Systeme das Ziel von Entwicklungsprojekten werden. Ob das mit UML, ereignisgesteuerten Prozessketten, Petri-Netzen, algebraischer Spezifikation oder sonstigen existierenden oder neu erfundenen Methoden passiert, ist egal. Hauptsache ist, sie werden so verwendbar gemacht und dann auch verwendet, dass Verifikation wirklich wird. Falsch taugt nicht.

Hans-Jörg Kreowski: Professor für Theoretische Informatik an der Universität Bremen, Vorsitzender des FIfF.