Warum Grundlagen lernen?

Viele Studenten der Informatik und verwandter Gebiete haben den Eindruck, daß das Grundlagenstudium und insbesondere die Beschäftigung mit theoretischer Informatik überflüssig sei und daß man die dafür aufgebrachte Zeit besser für Projekte, praktische Arbeiten und berufsnahe Studienthemen verwenden sollte.

Sie vergessen dabei jedoch, daß ein Studium an einer Universität mehr als nur eine Berufsausbildung ist – dafür wäre ein Studium an einer Fachhochschule oder Berufsakademie wesentlich besser geeignet – und daß gerade die nicht so spezifisch auf eine bestimmte Anwendung orientierten Grundlagenstudien einen in die Lage versetzen, sich schneller an die ständig ändernde Welt der IT anzupassen und grobe Fehler bei der Umsetzung einer Idee in konkrete Software zu vermeiden. Solche Fehler passieren aber schnell, wenn man das Leitziel der Theorie "beweise die Korrektheit der Resultate, die Du erzielst" außer acht läßt.

Ich will im folgenden illustrieren, wie das Mißachten der Grundlagen in der Praxis zu gravierenden Fehlern mit erheblichen Folgeschäden geführt hat und warum der gezielte Einsatz mathematisch-theoretischer Hilfsmittel oft zu erheblich besseren Lösungen führt, als die besten Programmierer ohne Verwendung theoretischer Einsichten erzielen können.

Kleine Fehler mit großen Konsequenzen

Wir wissen aus der theoretischen Informatik, daß man die Korrektheit von Software nicht testen kann und nicht darum herum kommt, diese mühsam zu beweisen, wenn es das Einsatzgebiet erforderlich macht. Dennoch glauben auch heute noch viele IT-Praktiker und selbst eine große Zahl von Forschern im Softwarebereich, daß intensives und systematisches Testen eine Garantie für fehlerfreie und zuverlässige Software bieten könne. Es stimmt, daß Testverfahren Fehler in Software aufspüren können und hierfür sind diese Verfahren auch unbedingt erforderlich. Eine Garantie aber können sie nicht liefern, denn im Gegensatz zu physikalischen Objekten, mit denen sich Ingenieure außerhalb des IT-Bereichs befassen, zeigt Software keinerlei stetiges Verhalten und somit ist eine Interpolation für den Bereich zwischen den Meßpunkten nicht zulässig - das Testergebnis für einen Meßpunkt sagt überhaupt nichts aus über das Verhalten der Software in der unmittelbaren Nachbarschaft.

Genau diese Diskrepanz hat in der Vergangenheit zu einigen groben Fehlern mit kostspieligen und zum Teil fatalen Konsequenzen geführt. Ich will einige dieser Pannen beispielhaft ansprechen.

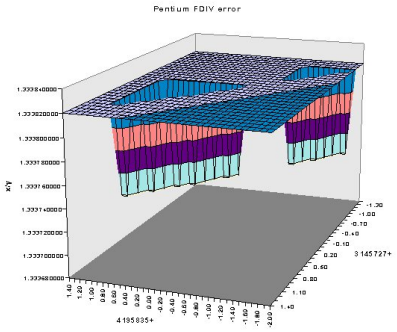
- Bei dem Absturz einer Boeing 767 im Mai 1991 hatte sich als Unfallursache herausgestellt, daß eines der Triebwerke während des Steigfluges auf Schubumkehr geschaltet hatte. Ingenieure des Airbus A320 nahmen dies zum Anlass, eine Sicherheitsschaltung für den Umkehrschub zu programmieren, die verhinderte, daß Umkehrschub eingeschaltet werden konnte, wenn sich sich die Räder des Landegestells nicht mit mindestens 70km/h drehen.

  Diese schnelle Lösung hatte fatale Konsequenzen bei der Landung eines Lufthansa Fluges in Warschau am 14. September 1993. Wegen starken Regens führte Aquaplaning auf der Landebahn dazu, daß sich die Räder nur geringfügig drehten und der Pilot die Maschine mangels Umkehrschub nicht rechtzeitig bremsen konnte. Die Maschine schoß über die Landebahn hinaus und ein Passagier und der Copilot kamen ums Leben.

  Hätte man die Frage "was definiert eigentlich eine Landung" gründlicher durchdacht anstatt vorschnell eine Lösung zu programmieren, wäre es nicht zu diesem Unfall gekommen.

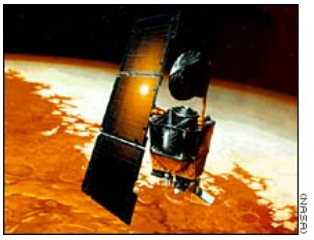
- Bei der Entwicklung des Pentium I Prozessors im Jahre 1994 wurde erstmalig ein neuer, deutlich schnellerer, Divisionsalgorithms verwendet. Trotz einer umfänglichen Serie von Tests beobachtete man kurz nach Auslieferung der Chips, daß der Prozessor bei manchen Divisionen falsche Ergebnisse lieferte. Wie sich herausstellte, waren die Hardwaretabellen des Algorithmus unvollständig in den Chipentwurf übertragen worden. Der Rückruf der 2 Millionen Prozessoren kostete die Firma Intel 450 Millionen Dollar.



- 

  Beim Jungfernflug der neuen ESA Ariane 501 Rakete im Juni 1996 löste der Steuerungscomputer 37 Sekunden nach dem Start eine Sprengung der Rakete aus, da ein Überlauf in der 16bit Arithmetik der eingesetzten Rechner zu unsinnigen Steuerbefehlen geführt hatte. Wie sich herausstellte, war die Schubkraft der neuen Rakete zu groß für eine Darstellung in 16bit Arithmetik. Nichtsdestotrotz war die Startsoftware unverändert von der Vorgängerrakete übernommen worden, da die Tests der Software keinerlei Auffälligkeiten zeigten. Der Schaden für die zerstörten Rakete und den beförderten Satelliten betrug mehr als eine Milliarde DM und ein Projekt mit Entwicklungskosten von 11.8 Milliarden DM wurde um mehrere Jahre zurückgeworfen.

- Im Jahre 1999 schickte die NASA zwei Sonden zum Mars, zuerst den Mars Climate Orbiter zur Erforschung der Marsatmosphäre und später den Mars Polar Lander zur Erforschung der Marsoberfläche. Der Orbiter sollte den Mars in einer Distanz von 160km umrunden, verschwand aber am 23. September plötzlich von der Bildfläche. Wie sich einige Zeit später herausstellte, hatte die Steuerungssoftware eine Umlaufbahn von unter 60km gewählt, da die Meßinstrumente Höhenangaben in Fuß lieferten, die Software aber davon ausging, daß die Angaben metrisch waren. Infolge der niedrigen Umlaufbahn wurde das Steuerungssystem überheizt und die Sonde im Wert von 150 Millionen Dollar stürzte ab.



  Drei Monate später erreichte der Mars Polar Lander die Marsoberfläche, zerschellte jedoch bei der Landung. Wieder waren Einheitenprobleme zwischen Meßinstrumenten und Steuerungssoftware die Ursache. Für eine saubere Landung muß die Abschaltung der Bremsraketen 40 Fuß (12 Meter) über dem Boden eingeleitet werden. Die Abschaltung begann jedoch schon bei 40 Metern über dem Boden, so daß die Sonde ungebremst aus 28 Meter Höhe abstürzte - ein Schaden von 165 Millionen Dollar und eine große Blamage für die amerikanische Raumfahrt.

- Der ISO/IEC 9796-2 Standard spezifiziert Algorithmen für digitale Signaturen af der Basis mathematischer Operationen. Der Sinn solcher Standards ist, daß Spezifikationen direkt in sicherheitskritische Anwendungen übertragen werden können. Bei einer Überprüfung der Spezifikation des sogenannten Legendre-Symbols mit einem Theorembeweiser im Jahre 2009 wurde jedoch festgestellt, daß die Spezifikation anstelle des notwendigen Kongruenzsymbols “==” ein Gleichheitssymbol “=” verwen-

dete. Dieser scheinbar unbedeutende Tippfehler hat zur Folge, daß die in kryptographischen Verfahren eingesetzten schnellen Primzahltests fehlschlagen und die Verschlüsselung unsicher wird. Der Fehler bestand seit mehr als 10 Jahren und ist wahrscheinlich ungeprüft in hunderte von Anwendungen übernommen worden.

Die Beispiele zeigen, daß die Bandbreite der möglichen Fehler von groben Vereinfachungen zu sehr subtilen Details reicht, die auch mit ausgefeilten Testverfahren kaum identifiziert werden können. Nichtsdestotrotz hätten alle Pannen vermieden werden können, wenn man wenigstens versucht hätte, die Zuverlässigkeit der eingesetzten Soft- und Hardware zu beweisen, anstatt sich nur auf die Ergebnisse der Tests zu verlassen.

Werkzeuge zur Unterstützung von Beweisführung

Nun sind Beweise etwas, mit dem sich Informatiker und Programmierer sehr ungerne beschäftigen. Beweise wirken im Verhältnis zur eigentlichen Entwicklungsaufgabe unproduktiv, sind gerade im Bereich Software wegen der Fülle der Details extrem mühsam und erinnern viele an die unangenehmen Teile ihres Studiums, deren Sinn sie nie so recht einsehen konnten. Aus diesem Grunde richtet sich ein Teil der Grundlagenforschung in der Informatik auf die Entwicklung von logischen Werkzeugen zur rechnergestützten Beweisführung. Die Bandbreite der Tools reicht von Typecheckern in Programmiersprachen, Model-Checkern für Hardware und Theorembeweisern für komplexe logische Probleme bis zu spezifische Verfahren zur Verifikation, Synthese und fehlerfreien Optimierung von Software.

Heute weiß man, daß der Fehler im Pentium Prozessor leicht mit einem Model-Checker hätte gefunden worden werden können, und überprüft deshalb jetzt jeden Hardwareentwurf mit diesem Werkzeug. Der Einsatz von Typecheckern hätte dazu geführt, daß die Einheitenfehler bei den Marssonden und vermutlich auch bei der Ariane-Steuerungssoftware schon bei der Programmierung aufgefallen wären, da man gezwungen worden ist, sich mit dem Thema Einheiten zu beschäftigen. Der Versuch einer Verifikation der Airbussoftware hätte die Ingenieure gezwungen, ihre problematische Definition des Begriff “Flugzeug ist gelandet” offenzulegen. Auch der Fehler im ISO Standard wurde nur aufgrund der Verwendung eines Theorembeweisers gefunden.

Darüber hinaus ist es mit formalen logischen Werkzeugen auch gelungen, systematisch Software aus logischen Spezifikationen zu entwickeln, die nicht nur garantiert korrekt, sondern auch um Größenordnungen effizienter ist als Algorithmen, die von Hand programmiert wurden. Dies wurde 1993 mit der Synthese von Scheduling Algorithmen für die US Air Force eindrucksvoll unter Beweis gestellt. Mit dem KIDS System konnte innerhalb von wenigen Stunden ein Algorithmus entwickelt werden, der mehr als 2000 mal schneller war als der existierende, in monatelanger Arbeit geschriebene ADA Code. Ein entscheidender Aspekt hierbei war, daß man sich auf die logische Struktur des Problems und konzeptuelle Optimierungen konzentrieren konnte, während die eigentliche Codierung samt Korrektheitsbeweis vom KIDS System erzeugt wurde.

Im Bereich Sicherheit sind heutzutage Grundlagen aus Mathematik, theoretischer Informatik und Algorithmen nicht mehr wegzudenken. Alle Verschlüsselungsverfahren, die man bis in die frühen 70er Jahre verwendet hatte, können mit Mitteln der Statistik und der linearen Algebra auf jedem PC innerhalb weniger Sekunden gebrochen werden. Moderne Sicherheitsmechanismen brauchen fundierte Kenntnisse in Gruppentheorie, Zahlentheorie, Informationstheorie und Komplexitätstheorie, um Informationen und Computer erfolgreich gegen Angriffe von außen zu schützen.

Auch außerhalb des Themenkomplexes Sicherheit und Zuverlässigkeit gibt es viele Anwendungen, die ohne Grundlagenarbeit nicht zum Erfolg kämen. So spielt die Komplexität von Suchalgorithmen eine entscheidende Rolle für die Effizienz von Graphikkarten und ihren Anwendungen – insbesondere in der Spie-

leprogrammierung. Ohne fundierte Kenntnisse aus den Grundlagen der Algorithmik und der Komplexitätstheorie kann man heutzutage keine konkurrenzfähigen Graphikanwendungen mehr schaffen.

Relevante Forschung und Lehre an der Universität Potsdam

Deswegen befasst sich der Lehrstuhl für Theoretische Informatik am Institut für Informatikmit der Entwicklung logischer Werkzeuge zum Entwurf sicherer und zuverlässiger Software. Unsere Forschungen richten sich auf verschiedene Aspekte

- *Entwicklung und Implementierung effizienter Theorembeweiser* für Logiken erster Stufe, insbesondere auch für konstruktive Logik und Modallogiken.
- *Verifikation, Synthese, und fehlerfreien Optimierung von Kommunikationssystemen.* Diese Forschungen werden mithilfe eines interaktiven Theorembeweisers in enger Kooperation mit dem Department of Computer Science der Cornell University in Ithaca, NY (USA) durchgeführt. Mit diesem Werkzeug konnte bereits ein subtiler Fehler in einem sorgfältig geprüften Kommunikationsprotokoll aufgespürt werden und die Performanz des implementierten Systems durch logische Optimierungen um den Faktor 10 zu steigern.
- *Synthese einer vollständig verifizierten elektronischen Chipkarte für US Bundesbehörden.* Bei diesem von der NSA gefördeten Projekt, das in enger Kooperation mit dem Kestrel Institute in Palo Alto, CA (USA) durchgeführt wird, geht es darum, den gesamten Code der SmartCard – vom Betriebssystem bis zu den kryptographischen Algorithmen – vollständig aus Spezifikationen zu entwickeln und mit einem Theorembeweiser zu verifizieren. Nur so kann das höchste Maß an Sicherheit garantiert werden, was für die vorgesehene Anwendung erforderlich ist.
- *Entwicklung theoretischer Grundlagen zur Formalisierung von Informationssicherheit.* Dieses noch relativ neue Thema knüpft an den Ende 2009 entstandenen DFG Forschungsschwerpunkt “Reliably Secure Software Systems” an und hat zum Ziel, Theorembeweiser für die Untersuchung von Sicherheitsaspekten nutzbar zu machen. Es ist bereits gelungen, erste Komponenten einer Sicherheitsvorschrift für Banken mit einem unserer Theorembeweiser zu verifizieren.

Die Arbeiten sind gekoppelt mit einem regelmäßigen Lehrveranstaltungsangebot zu Automatisierter Logik und Programmierung, Methoden des automatischen Beweisens, Kryptographie- und Komplexität und einer Reihe von Seminaren zu speziellen Aspekten unserer Forschung.

Christoph Kreitz ist Professor für Theoretische Informatik an der Universität Potsdam und Adjunct Professor am Department of Computer Science der Cornell University.