Anyone who has ever written a program knows how difficult it is to get even a small one to perform as intended. Often a program fails miserably the first time it is executed and begins to produce reasonable results only after several iterations of test and repair.

Most programmers are also familiar with the unpleasant experience of developing a program that works correctly for some time, perhaps years, and then fails. It is common for users to discover defects in fielded software that has been tested extensively.

In practice, most commercial software is very good, but still contains defects. It is not uncommon for a software release to deal primarily with fixing defects in earlier releases.

Even software that has been developed with great care for an important application has been known to fail. In 1981, the space shuttle software failed, at the time of the very first launch attempt, despite the tremendous care taken in its preparation. In 1990, the software that operates AT&T’s long-distance
Safety-critical software must perform as desired and should never fail. The need for dependability stems from the fact that the consequences of failure are extremely high, usually a threat to human life. To write such systems, most now agree that we must adopt rigorous techniques, rooted in mathematics.
SIGNPOSTS AND LANDMARKS: A SAFETY-CRITICAL READING LIST

Some conferences and journals are devoted in whole or in part to safety-critical computing. The types of problems that can occur with these systems are discussed each month in the Inside Risks column of Communications of the ACM. The annual Fault-Tolerant Computing Symposium is also a very long-running conference series and includes many papers on safety-critical software each year (IEEE Press). Other significant conferences are Dependable Computing for Critical Applications and the Conference on Computer Assurance (Compass).

Journals such as IEEE Transactions on Computers, IEEE Transactions on Reliability, and IEEE Transactions on Software Engineering regularly carry articles on safety-critical computing. For example, the January 1991 issue of IEEE Transactions on Software Engineering is devoted to software for critical systems and the May 1992 issue of the IEEE Transactions on Computers is devoted to fault-tolerant computing.

DEPENDABILITY ASSESSMENT.

TESTING. There is a large body of literature on testing, and interesting papers can be found in many journals and proceedings. The International Symposium on Software Testing and Analysis (known in earlier years as the Symposium on Testing, Analysis, and Verification) is an important focus for work on testing (ACM Press). The most recent proceedings were published in the July 1993 issue of ACM Software Engineering Notes. Wiley publishes the journal Software Testing, Verification and Reliability.

SAFETY. Software safety is an important topic in safety-critical computing. Two significant papers, both by Nancy Leveson, summarize the field and provide extensive bibliographies: “Software Safety: What, When, and How” (Computing Surveys, June 1986) and “Software Safety in Embedded Computer Systems” (Communications of the ACM, February 1991).

Various conferences and workshops have had software safety as either part or all of their theme. The 1991 Compass was concerned with systems integrity, software safety, and process security (IEEE Press). A workshop in September 1991 on Digital Systems Reliability and Nuclear Safety, sponsored by the US Nuclear Regulatory Commission and the National Institute of Standards and Technology, addressed a variety of safety issues surrounding the use of digital systems in nuclear applications (NIST, Washington DC, to appear).

— John Knight and Bev Littlewood

system failed, despite a great deal of quality control and years of successful operation prior to the failure.

Writing software is difficult.

DIRE CONSEQUENCES

Commercial software must be dependable, but the consequences of failure are usually relatively small. A spreadsheet program that does not display a graph correctly or causes a machine to lock up is more of an irritant than anything else.

In safety-critical applications, on the other hand, the consequences of failure are extremely high, usually a threat to human life. Safety-critical systems are expected to perform as desired and never fail.

There are many well-known safety-critical applications in areas like transportation, defense, health care, and nuclear-power generation. In transportation, for example, computerized train-control and -signaling systems are safety critical because failure could lead to serious accidents.

Beyond these well-known examples, many other applications are emerging where the consequences of failure are high but not obviously life-threatening. The loss of valuable equipment or money can be so serious that applications in which this can occur might have to meet very high dependability criteria, although they might not be routinely described as safety critical. For example, loss of tele-
phone service has a significant societal impact because of the inability to place emergency calls.

When a computer system can cause these kinds of catastrophic failure, developers and regulators will apply very high dependability criteria to both the hardware and the software.

DEMONSTRATING QUALITY

To address the problems posed by safety-critical applications, the goal must be stated carefully. It is not sufficient to state that a particular software system must be "very reliable" or "defect free." In practice, the word "reliability" is often used when the real goal is availability or safety, and adjectives such as "very" are too imprecise.

One way to state the goal is to use probabilities. Reliability, for example, is a well-defined term in engineering that can be applied to systems that contain software. A system's reliability is stated as a function of time and is defined as the probability that the system will operate correctly in a specified environment for the stated time. Similarly, availability can be expressed as the probability that a system is operating correctly at a specified time. In many safety-critical-application areas, regulating agencies specify such probabilities.

So the goal of engineers who build safety-critical systems is to meet some quality criteria that will very likely be stated as a probability. The first problem is to achieve such quality. But a second and extremely important problem is to demonstrate that the goal has been achieved.

For some applications, the dependability requirements can be fairly modest. In certain medical contexts, for example, the alternative to computers is people, and people are known to be highly fallible. Usually, however, safety-critical systems can fail only infrequently at most.

In these cases, it is very difficult to obtain sufficient data — by observing no failures or an acceptable number of failures over a very extended time — to demonstrate that a quality goal has been reached. It is very possible that a software system actually meets a quality goal, but it is impossible to demonstrate that it does.

When faced with software failures, many engineers suspect inadequate testing. Unfortunately, traditional software-development techniques usually do not provide the levels of dependability demanded by safety-critical systems, and the quality criteria are usually such that the amount of testing that is feasible cannot demonstrate that the desired goals have been achieved.

The technique known as proof-of-correctness is sometimes suggested as a solution to the problem of building safety-critical software. This technique involves establishing a proof of correspondence between a software system's specification and the software itself, usually written in a high-level programming language. A proof that the software will terminate is also required.

The idea that a proof exists that shows software to be correct is appealing, but very misleading. Proof-of-correctness is a technical term that does not mean what it appears to mean. It does not account for the possibility that the specification, or the proof itself, might be wrong. There are many other practical problems with this technique, such as dealing with compilers and other system software.

Nevertheless, it is generally agreed that large improvements in developing dependable software are most likely to be achieved by applying mathematics in some form. The result might not be perfection, as implied by proof-of-correctness, but practical techniques that yield solid gains in dependability.

Software-development techniques based on mathematics are called formal methods, and they are beginning to become both practical to apply and capable of providing substantial improvements in software quality. For example, formal methods are being applied very successfully to specifications. Notations for software specifications that have very precise meanings and can be manipulated by computer — such as Z, VDM, and Statecharts — are being used to develop substantial software systems.

THE IDEA THAT A PROOF EXISTS THAT SHOWS SOFTWARE TO BE CORRECT IS VERY APPEALING, BUT VERY MISLEADING.

Because of the increasing awareness of and interest in the use of formal methods to develop safety-critical software, there is a definite formal-methods flavor to this issue. This editorial slant was not caused by any specific selection process on our part, but many of the articles submitted for this special issue dealt with subjects related to formal methods. This is a significant endorsement of formal techniques by the safety-critical community.

In the first article, Susan Gerhart, Dan Craigen, and Ted Ralston summarize their important study of 12 applications that used formal methods to build safety-critical systems. The authors concentrate on four systems that had external regulatory requirements: Darlington Nuclear Generating Station, Paris Metro, Traffic Alert and Collision-Avoidance System, and Multinet Gateway System. Each of these applications is described briefly in case studies that follow the main article.

The second article also involves a regulatory application. Ramin Mojdehbaksh and colleagues report their work to satisfy the strict requirements of the US Food and Drug Administration to demonstrate that existing products are safe. They describe a way to retrofit safety through a rigorous software-safety analysis to identify safety faults, modify the software to mitigate the faults, and verify overall safety. Although the application involved an implantable medical device, much of the process can be applied to products in other domains.

Validating specifications against functional and safety requirements remains one of the more difficult aspects of critical-system development. In the third article, Lloyd Williams compares specifications for a water-level monitor written using a version of the Software Cost Reduction methodology with those written using the Vienna Definition. He also identifies key assess-
ment issues and compares various assessment techniques.

Finally, Wolfgang Halang and Bernd Krüger look at using formal specification and validation techniques for a particular class of safety-critical systems: programmable logic controllers. They describe how to create function blocks, demonstrate their correctness, and interconnect them to make safety verification easier.

These articles cover a wide variety of topics and perspectives that range from research to practice. We hope you will find their presentation alongside each other useful, interesting, and stimulating.

ACKNOWLEDGEMENTS

We thank the many referees and all the authors who submitted articles to this special issue.

John Knight is a professor of computer science at the University of Virginia. His interests are software dependability, with an emphasis on safety-critical systems. His research program includes projects on the rigorous development of formal specifications, operating-system support for safe software operation, dependable implementation techniques, and the exploitation of software reuse to increase software quality.

Knight received a BS, with honors in mathematics from Imperial College, London, and a PhD in computer science from the University of Newcastle-upon-Tyne.

Bev Littlewood is a professor of software engineering and director of the Centre for Software Reliability at City University. His interests include modeling and evaluating reliability, safety, and security of software.

Littlewood has degrees in mathematics and statistics and a PhD in computer science and statistics. He serves on the UK government’s Advisory Committee on the Safety of Nuclear Installations, the IFIP working group on reliable computing and fault tolerance, and the British Computer Society’s Safety-Critical Systems Task Force. He serves on the editorial boards of several journals, including IEEE Transactions on Software Engineering.

Address questions about this issue to Knight at Computer Science Dept., University of Virginia, Thornton Hall, Charlottesville, VA 22903; knight@virginia.edu or to Littlewood at the Centre for Software Reliability, City University, London, E14 0H, UK, b.littlewood@csr.city.ac.uk.

DISTANCE EDUCATION

LEADING TO A MASTER OF SCIENCE IN
SOFTWARE ENGINEERING

OFFERED BY
THE SCHOOL OF ENGINEERING
AND APPLIED SCIENCE AT
SOUTHERN METHODIST UNIVERSITY

The program is offered via VIDEO TAPE nationwide as well as on the SMU campus.

The program is centered about the problems facing the working professional in the field. Emphasis is on both the fundamental principles of software and system design, and the practical problems of commercialization. Much of the subject matter of the core is based on the curriculum recommended by the Software Engineering Institute. The elective courses enable the student to focus on particular interests.

ADMISSION REQUIREMENTS: BS/BA in one of the sciences or engineering disciplines with a 3.0/4.0 GPA and one year experience in software development or maintenance.

For complete information contact:
Michael Kirkpatrick
Southern Methodist University
(214) 768-1452

SMU does not discriminate on the basis of race, color, national or ethnic origin, sex, or disability.

NORTHROP
Electronics Systems Division
Rolling Meadows Site

NORTHROP
Electronics Systems Division
Rolling Meadows Site