Software reliability: modern challenges

Diomidis Spinellis
Department of Information and Communication Systems, University of the Aegean, Greece

ABSTRACT: The evolution of computer technology is creating for safety-critical systems new challenges and different types of failure modes. Modern computer processors are often delivered with errors, while intelligent hardware subsystems may exhibit nondeterministic behaviour. Operating systems and programming languages are becoming increasingly complicated and their implementations less trustworthy. In addition, component-based multi-tier software system architectures exponentially increase the number of failure modes, while Internet connectivity exposes systems to malicious attacks. Finally, IT outsourcing and blind reliance on standards can provide developers with a false sense of security. Planning in advance for the new challenges is as important as embracing the new technology.

In G. I. Schuëller and P. Kafka, editors, *Proceedings ESREL '99 — The Tenth European Conference on Safety and Reliability*, pages 589–592, Munich-Garching, Germany, September 1999. ESRA, VDI, TUM, A. A. Balkema.

1 INTRODUCTION

Software increasingly forms a critical part in the design and operation of products, processes, equipment, and installations affecting their safety and reliability (Winter et al. 1998). Despite the important advances made over the last decades in the area of software engineering and the successful realisation of many safety-critical software systems, the evolution of computer technology is creating new challenges and different types of failure modes. In the following sections we examine how advances and changes in the areas of computer hardware components and subsystems, operating systems, software system architectures, programming languages, and the software development process can potentially affect the safety and reliability of computer-based systems. Despite the luddite connotations of our presentation we believe that a critical examination and appraisal of these advances and their effects is of paramount importance in the area of safety-critical applications.

2 HARDWARE

2.1 Components

Increasing chip densities have resulted in significant advances in processor hardware performance based on large scale integration of functional units,

Processor	Errata
Intel 80C186 Embedded Processor	5
Texas Instruments TMS320C40 DSP	17
MIPS R4000SC	55
MIPS R4400PC/SC	23
Intel 386 EX Embedded Processor	40
Intel Pentium	82
Intel Pentium Pro	77
Intel Pentium II	58

Table 1: Documented errors in processor implementations.

pipelined designs, and the provision of additional functionality (Hennessy and Patterson 1990, pp. 250–349). However, as a result of pipelined architectures, on modern processors it is practically impossible to reason accurately and completely about the execution of a program at the lowest level. The interdependencies of the multiple functional units, many levels of cache, and branch predictors all dynamically changing their behaviour as the program executes — often in a multitasking environment — make the isolation of problems that can arise at this level an experimental rather than analytical reasoning task. Safety critical software systems operating in such an environment can not therefore be proven to satisfy a specification using formal methods.

In addition, as the complexity of processors increases so do the errors that are part of a given implementation. Compiler and system software writers, but often also end-user software developers have to be aware of those errors and design their implementations around all *known* errors. Some of these hardware errors can result in a complete system lock, others in data corruption, and others in subtle differences in arithmetic results; obviously all of them important to the designer of a computer-based safety-critical system. Table 1 illustrates the number of different errors documented (e.g. (Intel Corp. 1996)) in some processor implementations.

2.2 Subsystems

Modern microprocessor-controlled components such as disk drives, network adapters, and graphic controllers often contain enough intelligence to create a potential for problems at the system integration level. As an example, many modern hard disks rely on a thermal recalibration procedure to compensate against temperature-induced changes in the drive's physical characteristics. Under some circumstances, an unsophisticated implementation of this procedure can delay the drive's response time at random instances rendering it unsuitable for real-time applications (Taylor et al. 1995).

Operating System	Year	Number of system calls
First Edition Unix	1971	33
Seventh Edition Unix	1979	47
SunOS 4.1	1989	171
4.3 BSD Net 2	1991	136
HP-UX 9.05	1992	219
SunOS 5.4	1994	163
Linux 1.2	1996	211
SunOS 5.6	1997	190
Linux 2.0	1998	229
Windows Platform SDK	1998	3433

Table 2: Increasing operating system complexity as demonstrated by the number of supported system calls.

The allocation of interrupts and input/output addresses in PCI-based "Plug & play" systems is performed at system startup using a complicated negotiation procedure among active subsystem components. In Windows-based systems supporting drivers and modules are then loaded and run in a nondeterministic order. As a result, Gutmann (1998) reports that the state of such systems after a reboot is relatively unpredictable. The implication of this is that the establishment of a stable test platform or the reproducibility of faults following a specific line of actions may not be feasible.

3 OPERATING SYSTEMS

Operating systems are constantly increasing in complexity. A rough measure of their complexity can be drawn by examining the number of supported system calls illustrated in Table 2. A system call de-

fines an interface to the operating system; more system calls increase the complexity of the operating system needed to support them, provide additional opportunities for unwanted interactions between them, and increase the chances of overlooked security loopholes.

This increasing complexity has important implications for the reliability of software developed for a specific platform. Complicated interfaces are difficult to learn and use effectively (Spinellis 1998b). As a result of their size and complexity, modern operating systems exhibit an increasing number of bugs; demonstrated by the numerous "fixes" distributed by their vendors. Developers of robust applications have to take this into account coding around them, or insist on the installation of all relevant fixes. Some fixes may even introduce new errors or render other system components inoperative. The bottom-line of this situation is, that the application developer is practically rarely singly responsible for the reliability of an application.

4 SOFTWARE SYSTEM ARCHITECTURE

Modern networked, multi-tier software system architectures exponentially increase the number of failure modes based on the number of interconnected nodes (Spinellis 1998a). Software commercial-of-the-shelf (COTS) components are increasingly used as parts of integrated systems (Voas 1998b). Their quality is often difficult to assess (Voas 1998a) and due to the tight coupling between components enforced by some programming languages (structured exception handling, heap-based dynamic memory allocation, and unbounded pointers) they may affect the reliability of the software system in totally unforeseen ways.

The use of the Internet as a common network infrastructure often exposes applications to additional failure modes related to the open and insecure nature of the medium. Applications using the Internet as a data pipe can face problems related to connectivity, congestion, routing, and the domain name system. In addition, such applications are exposed to hostile attacks that can be carried out over the network (Bhinami 1996; Denning 1990). Typical applications are not coded to guard against malicious attacks; in fact even system software that should have been coded in such a way is often compromised (Spafford 1989). Therefore, the connection of any safety-critical system to the Internet can severely affect its reliability.

Title	Year	Pages
The C Programming Language (Kernighan and Ritchie)	1978	228
The C Programming Language; second edition (Kernighan and Ritchie)	1988	272
The C++ Programming Language (Stroustrup)	1986	328
The C++ Programming Language; second edition (Stroustrup)	1991	669
The C++ Programming Language; third edition (Stroustrup)	1997	910

Table 3: The evolution in complexity of C and C++.

5 PROGRAMMING LANGUAGES

Similarly to operating systems, programming languages also have a tendency to grow in size and complexity as they mature. Taking as a rough measure the page number of the language's canonical description Table 3 provides an illustration of the evolution of the C and C++ programming languages.

This trend has important implications for the developers of high-reliability systems. Large languages are difficult to learn and use (Hoare 1983). It is nowadays not uncommon for programming teams to lack people who understand the whole language at a level sufficient to advise other members on issues regarding the interrelationship between language elements. Subtle bugs arising from the misunderstanding of language features can thus survive code walkthroughs. In addition, language complexity and advanced optimisation techniques combined with processor complexity results in an increased number of bugs in modern compilers. This is clearly an additional risk factor for high-reliability designs.

6 SOFTWARE DEVELOPMENT PROCESS

The changing nature of the software development process can also negatively affect the reliability of the delivered system. Information technology outsourcing (Lacity et al. 1995) may exclude the contractor's software developers from a holistic system-wide perspective resulting in dangerous misunderstandings and grey areas of responsibility. In addition, the increasing adoption of quality systems such as the ISO-9000 series (International Organization for Standardization 1991) and the Capability Maturity Model (Herbsleb et al. 1997) may provide software developers and procurers with a false sense of security.

7 CONCLUSIONS

Despite the advances made over the last decades in the design and implementation of safety-critical systems, major new challenges lie ahead. It is important for managers, designers, and developers to be aware that all architectural, technological, and organisational improvements in the realisation of software systems carry with them new challenges and dangers. Their solution is most probably not technology-based (Brooks, Jr. 1987). In the demanding world of software-based safety-critical systems planning in advance for the new challenges is as important as embracing the new technologies.

Acknowledgements

The work reported herein was carried out within the context of ISA-EUNET, an ESPRIT (ESSI-ESBNET, project number 27450) R&D project funded by the Directorate General III of the European Commission.

REFERENCES

Bhinami, A. (1996, June). Securing the commercial internet. *Communications of the ACM 39*(6), 29–35.

Brooks, Jr., F. P. (1987, April). No silver bullet: Essence and accidents of software engineering. *IEEE Computer 20*(4), 10–19.

Denning, P. J. (1990). *Computers Under Attack: Intruders, Worms, and Viruses*. Addison-Wesley.

Gutmann, P. (1998, January). Software generation of practically strong random numbers. In *7th USENIX Security Symposium*, San Antonio, TX, USA. USENIX Association.

Hennessy, J. L. and D. A. Patterson (1990). *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers.

Herbsleb, J., D. Zubrow, D. Goldenson, W. Hayes, and M. Paulk (1997, June). Software quality and the capability maturity model. *Communications of the ACM 40*(6), 30–40.

Hoare, C. A. R. (1983). Hints on programming language design. In E. Horowitz (Ed.), *Programming Languages: A Grand Tour*, pp. 31–40. Computer Science Press. Reprinted from Sigact/Sigplan Symposium on Principles of Programming Languages, October 1973.

Intel Corp. (1996, July). 80C186 and 80C188 embedded microprocessors specification update. Online. http://www.intel.se/design/-intarch/specupdt/27289401.pdf. 29 August 1998.

International Organization for Standardization (1991). *Quality management and quality assurance standards — Part 3: Guidelines for the application of ISO 9001 to the development, supply and mantenance of software*. Geneva, Switzerland: International Organization for Standardization. ISO 9000-3:1991(E).

Lacity, M. C., L. P. Willcocks, and D. F. Feeny (1995, May-June). IT outsourcing: Maximize

flexibility and control. *Harvard Business Review 73*(3), 84–93.

Spafford, E. H. (1989, June). The Internet worm: Crisis and aftermath. *Communications of the ACM 32*(6), 678–687.

Spinellis, D. (1998a, November/December). The computer's new clothes. *IEEE Software 15*(6), 14–17.

Spinellis, D. (1998b, November). A critique of the Windows application programming interface. *Computer Standards & Interfaces 20*, 1–8.

Taylor, H., D. Chin, S. Knight, and J. Kaba (1995, Summer). The magic video-on-demand server and real-time simulation system. *IEEE Parallel & Distributed Technology 3*(2), 40–51.

Voas, J. M. (1998a, June). Certifying off-the-shelf software components. *Computer 31*(6), 53–59.

Voas, J. M. (1998b, June). The challenges of using COTS software in component-based development. *Computer 31*(6), 44–45.

Winter, V. L., J. M. Covan, L. J. Dalton, L. Alkalai, A. T. Tai, R. Harper, B. Flahive, W.-T. Tsai, R. Mojdehbakhsh, S. Rayadurgam, K. Mori, and M. R. Lowry (1998, April). Key applications for high-assurance systems. *Computer 31*(4), 35–45.