Formal Methods and Software Reliability

Gerard J. Holzmann
JPL Laboratory for Reliable Software
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91006

Abstract

In this position statement I briefly describe how the software reliability problem has changed over the years, and the primary reasons for the recent creation of the Laboratory for Reliable Software at JPL.

1. Introduction

An often noted trend is that as computers are becoming more powerful, continuing to follow the trend predicted by Gordon Moore some forty years ago [6], typical software applications tend to grow in size and complexity, and are therefore becoming harder to analyze thoroughly.

Although code size does not correlate too well with code complexity, it is one of the simplest metrics available and can therefore easily be used to confirm at least part of the postulated trend.

The size of OS/360 was given in 1968 as 5 Million lines of assembly level code (corresponding to approximately 1 Million lines of C code) [7]. Today's commercial operating systems are usually estimated to have around 20 Million lines of source code, which gives an increase of roughly **two** orders of magnitude since 1968. Another example, closer to home, is for software analysis systems. The size of *trace*, the earliest predecessor of SPIN [4], in 1983 was 3,500 lines of C, while the SPIN sources today stand at close to 30,000 lines of C. The increase is **one** order of magnitude this time, since 1983.

At JPL we can also track the evolution of flight software for interplanetary spacecraft over a long period of time. Doing so largely confirms these trends. From the Voyager spacecraft in 1977 to the Mars exploration spacecraft from 2004, the average size of flight software has, for instance, increased from approximately 4,000 lines of non-comment source to roughly 400,000 lines; an increase of **two** orders of magnitude since 1977.

Though by itself impressive, this rate of increase is well below the rate of increase in the speed of computers in the same period of time. Following Moore's law, the increase of raw computer power has been almost **four** orders of magnitude (2^{13}) since 1983, more than **five** orders of magnitude (2^{18}) since 1977, and approaching **seven** orders of magnitude (2^{23}) since 1968. This means that, by a large margin, computers are growing in power faster than programs grow in size. As confirmation of this trend, it can, for instance, readily be noted that a compiler can today process an average size program much faster, and produce better quality code, than a compiler could do in 1968 for the much smaller average size program from then.

Unfortunately, for software analysis purposes, complexity is not just determined by program size.

2. Complexity and Reliability

Not just the size but also the basic structure of a typical software application has changed over the years. Where in the sixties software applications were mostly sequentially executing, standalone programs, today both commercial software and spacecraft software applications are typically designed as multi-threaded, reactive systems: the behavior of these systems is not just determined by the response to a fixed set of input data, it also depends on the time of arrival of the inputs, and the many subtleties of interleaved process execution.

In general, any sufficiently interesting property of even a basic, non-reactive, deterministic computer program (e.g., halting) is formally undecidable [9]. But this well-known fact does not doom all attempts to perform some form of program analysis. If, for instance, we fix the input, make the program strictly finite state (e.g., by bounding the maximum amount of memory that may be used) most problems of interest do become decidable (including halting).

If all software applications would have remained in this simple class of non-reactive, deterministic, closed and finite-state systems, sophisticated program analysis methods might have completely replaced *ad hoc* testing techniques by now. But real-life is much more interesting than that. The non-determinism of a multi-threaded system can increase the complexity of program verification by an exponential amount, potentially using up all the gains made by the similarly exponential increase in compute power based on Moore's law. A modern spacecraft system, for instance, typically maintains over fifty concurrent threads of execution, as part of its basic behavior.

The conclusion of these observations is not that the battle with software complexity cannot be won. The conclusion is merely that if we cannot win the battle for software reliability by exploiting the brute force of Moore's law, we will have to become *smarter* about how we design, analyze, and test software systems. All these trends strengthen the opportunity for the further development, and application, of formal methods. These observations have provided the primary motivation for the recent creation of a new Laboratory for Reliable Software (LARS) at JPL.

3. The JPL Laboratory for Reliable Software

The role of LARS [10] is to find ways to increase the reliability of both flight and ground software for space missions by the application of state-of-the-art theories, techniques, and tools, where necessary developing new theory and harnessing them in tools that can be applied in the software development process. Within the scope of the new lab are improvements in *defect insertion* methods in the design phase of a project, as well as improvements in *defect detection* methods based on design and software analysis.

LARS builds on the success of some early application of logic model checking techniques to flight software systems to detect design problems, as described in, for instance, [1,3,5,8]. The scope of LARS, though, is significantly broader than applications of logic model checking techniques. One of the early focus points for LARS is to evaluate and further develop static source code analysis techniques, by looking at the leading commercial (www.polyspace.com, www.coverity.com) and academic source code analysis tools, including also [2] as a minor contender. LARS also targets improvements in early fault detection techniques, for instance by the development of improved requirements capture and analysis techniques, and similarly it will investigate advanced run-time monitoring techniques in collaboration with colleagues from NASA Ames Research Center.

As in most software systems, in spacecraft software even a small coding error can have large consequences. Unlike most earthly applications, though, the crash of a software controller on a spacecraft that operates millions of miles away from earth, could easily prove to be fatal, leading to a complete loss of the spacecraft, or a significant loss of the science return from a mission. Reliable software, therefore, is not a negotiable option at JPL, that can be weighed against market pressures; it is a firm requirement. It is the purpose of LARS to help JPL find, where possible, or develop, where necessary, the best available techniques that can secure truly reliable software systems.

Acknowledgements

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

References

1. P.R. Gluck, G.J. Holzmann, Using Spin Model Checking for Flight Software Verification, *Proc. Aerospace Conference*, IEEE, Big Sky, MT, USA, March 2002.
2. G.J. Holzmann, Static source code checking for user-defined properties, *Proc. Integrated Design and Process Technology (IDPT)*, Pasadena CA USA, June 2002.
3. G.J. Holzmann and R. Joshi, Model-driven software verification, *Proc. 11th Spin Workshop*, Barcelona, Spain, April 2004. Springer Verlag, LNCS 2989, pp. 77-92.
4. G.J. Holzmann, *The Spin Model Checker: primer and reference manual*, Addison-Wesley, 2004.
5. E. Mikk, P. Pingree, G.J. Holzmann, D. Dams, and M.H. Smith, Validation of mission critical software design and implementation using model checking. *Proc. 21st Digital Avionics Systems Conference*, IEEE, 27-31 Oct. 2002, Irvine, California.
6. G. Moore, Cramming more components onto integrated circuits, *Electronics Magazine*, Vol. 38, 19 April 1965, pp. 114-117.
7. P. Naur and B. Randell, (Eds.), *Software Engineering: Report of a conference sponsored by the NATO Science Committee*, Garmisch, Germany, 7-11 Oct. 1968, Brussels, 231 pp.
8. F. Schneider, S.M. Easterbrook, J.R. Callahan, and G.J. Holzmann, Validating Requirements for Fault Tolerant Systems using Model Checking, *Proc. Int. Conference on Requirements Engineering (ICRE)*, pp. 4-14, IEEE, Colorado Springs Co. USA, April 1998.
9. A.M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Mathematical Soc.*, Ser. 2-42, 1936, pp. 230-265.
10. URL: http://eis.jpl.nasa.gov/lars