# The Verified Software Repository

Jim Woodcock
University of York, United Kingdom

Industrial software usually has extensive documentation, but software behaviour often comes as a complete surprise. Depending on the circumstances, these surprises can range from the disastrous to the merely annoying. Why should software not behave as expected? Programming is inherently hard because we have to reduce a problem to a set of rules that can be blindly followed by a computer. This process is error prone, and this is compounded by components interacting and interfering, and undesirable properties emerging. The result is that systems often fail to satisfy their users' needs.

Software is also inherently hard to develop. It's hard to define requirements, to anticipate interactions, and to accommodate new functionality. Documentation involves large amounts of text, pictures, and diagrams, but these are often imprecise and ambiguous. Important information is often hidden by irrelevant detail. Design mistakes are often discovered too late, making them expensive or even impossible to correct. It's a tribute to the skill of software engineers that systems work at all.

Some practitioners in industry and researchers from universities believe it's now practical to use formal methods to produce software, even non-critical software. And that this will turn out to be the cheapest way to do it. Given the right computer-based tools, the use of formal methods could become widespread and transform the industrial practice of software engineering. The computer science community recently committed itself to making this a reality within the next fifteen to twenty years. Recent advances in theory and tool support have inspired industrial and academic researchers to join together in an international Grand Challenge (GC) in Verified Software to make its objectives a reality.

Our Grand Challenge has two central principles: theory should be embodied in tools; and tools should be tested against real systems. These principles are embodied in our two main current activities: the creation of a strong software engineering tool-set; and the collection of an appropriate range of examples. The challenge is to achieve a significant body of verified programs that have precise external specifications, complete internal specifications, and machine-checked proofs of correctness with respect to a sound theory of programming.

A Scientific Repository is a selected collection of scientific data constantly accumulating from experimental or observational sources, together with an evolving library of programs that process the data. The Verified Software Repository is a scientific repository serving the advancement of Computer Science, particularly in the area of strong software engineering and the mechanical certification of computer programs. The long-term goal of the repository is to develop a toolset of software engineering aids that could be commercialised for use in the construction, deployment, and continuous evolution of dependable computer systems.

These lectures describe experiments in verifying different kinds of operating system components. We draw conclusions about the current state of the art in modelling and verification, and the need for inter-operability of notations and tools. We finish by proposing a series of open challenge problems.

**References**


1. J. Woodcock and J. Davies. *Using ZSpecification, Refinement, and Proof*. Prentice Hall International Series in Computer Science; 1392pp; 1996.

2. J. Woodcock, S. Stepney, D. Cooper, J. Clark, and J. Jacob. *The Certification of the Mondex Electronic Purse to ITSEC Level E6*. Formal Aspects of Computing 20(1); 2008.

3. L. Freitas and J. Woodcock. *Mechanising Mondex with Z/EVES*. Formal Aspects of Computing Journal 20(1); 2008.

4. J. Woodcock. *First Steps in the Verified Software Grand Challenge*. IEEE Computer 39(10); pp 57–64; 2006.