Trustworthy Software:
Why we need it, Why we don’t have it, How we can get it

John McLean
Information Technology Division
Naval Research Laboratory
Washington, DC 20375-5337
John.McLean@nrl.navy.mil

We have become increasingly dependent on a national technological fabric that contains software, computers, and communication networks as essential components. Unfortunately, our ability to build affordable software systems for which there exists compelling evidence that the systems delivers their services in a manner that satisfies certain critical properties has not kept pace with the importance that these systems play in our lives. I think that there are five basic research issues that I think we must address if we are to advance our ability to build trustworthy software.

1. Formal methods that have a predictable, quantifiable impact on validation.

In many areas we have made great strides in producing trustworthy software. A notable example within the computer security community is the use of formal methods to analyze cryptographic protocols. Despite such successes, however, we have not seen formal methods become part of the standard software development process even though there is evidence that they increase software quality and that they can reduce over-all system cost. One reason is that even if formal methods increase system quality, it is hard to quantify this increase. For example, within computer security a full cost-benefit justification for applying formal methods would require that we could calculate the increased expense one would incur by breaking into the system and the value of the assets protected by the system. In general, neither figure is easy to calculate. However, it would be feasible to quantify the reduction, e.g., in testing that could result from applying formal methods. A current problem with many formal security proofs is that they fail to eliminate any particular class of system tests. Instead, they tend to increase comfort across the board. We must move away from this “across the board” approach to an approach where formal methods eliminate the need for considering specific fault scenarios.

2. Formal methods that support flexible release policies.

A second reason formal methods haven’t caught on is their impact on development. It’s not that formal methods cost more money and take more time than other software development paradigms; there is evidence that they reduce cost in the long run and actually lead to faster development of the final product. However, formal methods require one to spend more time in up-front analysis and push back code development until the end of the development cycle. In an environment where time to market can be the most important factor in determining success, it is a better strategy to produce something early that can be pushed out the door at short notice, if necessary, than to be tied to a rigid schedule that won’t produce a workable system until the very end. Even if low quality of a premature release leads to expensive fixes later on, this will be more than offset if early release helps the product become a de facto standard to which everybody else must build.

3. Improved methods for refinement and composition for all properties of interest.

A third problem with current techniques for building trustworthy systems is their inability to accommodate large-scale development practice. By large-scale development, I don’t mean development of systems with lots of code, although there is undoubtedly a relationship. I’m more focused on developments that require a high level specification that must be refined into numerous lower level specifications that will eventually be implemented into a set of components that will compose the ultimate system. It is well known that most properties are preserved by refinement and composition. It is well known in the security community that many properties we are interested in are preserved neither by refinement nor composition. (See, for
example, [1].) If we are to use either top down or compositional development, these limitations must be addressed.

4. Methods for building systems that must satisfy multiple properties.

Current techniques for building trusted systems focus on individual properties. Trustworthy software is required in a variety of systems – military, aviation (including flight control and air traffic control), transportation, financial, medical, SCADA etc - and must satisfy a variety of properties, including security, safety, real-time, and fault-tolerance. Unfortunately, each of these communities have their own modeling techniques, which makes it hard to reason about any system that must satisfy more than one property. Worse, many of the techniques required to enforce one property undermine enforcement of others. The security community, for example, tends to use nondeterminism; the real-time community tends to avoid it. We need techniques that support the development of systems that must satisfy multiple properties.

5. Building trustworthy software from untrusted components.

A final problem is that we cannot afford to verify everything, and many of our systems will end up containing code in which we can have no trust at all. Cost considerations will force us to use COTS, and off-shore software development make COTS use in certain systems suspect, at best. Perhaps, our biggest challenge will be to determine how to build trustworthy systems that contain numerous untrusted components.

References


Dr. John D. McLean is Superintendent of the Naval Research Laboratory’s Information Technology Division, where he previously served as Director of the Division’s Center for High Assurance Computer Systems and NRL’s Senior Scientist for Information Assurance. While at NRL he has held positions as an Adjunct Professor of Computer Science at the University of Maryland, the National Cryptologic School, and Troiseme Cycle Romand d’Informatique and as a Senior Research Fellow at the University of Cambridge’s Centre for Communications Systems Research.