Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 238 (2009) 11–17
www.elsevier.com/locate/entcs

A Software Certification Consortium and its Top 9 Hurdles

John Hatcliff[a,1], Mats Heimdahl[b,2], Mark Lawford[c,3], Tom Maibaum[c,3], Alan Wassyng[c,3,4], Fred Wurden[d,5]

[a] *Department of Computing and Information Sciences, Kansas State University, Manhattan, KS, USA*

[b] *U of Minnesota Software Engineering Center, Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA*

[c] *Software Quality Research Lab, McMaster University, Hamilton, ON, Canada*

[d] *Microsoft Corp., Seattle, WA, USA*

Abstract

In August of 2007 and December of 2007, North American academic researchers, industry representatives and regulators were invited to meetings in Washington and Minneapolis, respectively, with the goal of forming a Software Certification Consortium (SCC). At the first meeting, objectives were established for the consortium and a certification grand challenge was issued. At the second meeting, all participants were asked to complete the statement: "Software certification is hard because . . .". The group then synthesized the results into a "Top 9" list by means of discussion and voting. In this article, we describe the goals that we believe should be the goals of SCC, via details of these Top 9 hurdles that are preventing us from making software certification part of the mainstream.

Keywords: Software Certification Consortium (SCC), Objectives, Projects, Formal Methods

1 Introduction

People are increasingly dependent upon software in their daily lives. In addition to all the conventional software driven systems we are familiar with in our offices and homes, software in embedded systems implements the control algorithm in anti-lock brakes in our cars, fly-by-wire systems in airplanes, nuclear power plants, and life saving biomedical systems. All of these systems are "hard real-time" systems that must react with precise timing properties to function correctly. In an effort

[1] Email: hatcliff@cis.ksu.edu
[2] Email: heimdahl@cs.umn.edu
[3] This work was partially supported by NSERC
[4] Email: lawford,maibaum,wassyng@mcmaster.ca
[5] Email: fredwurd@microsoft.com

doi:10.1016/j.entcs.2009.09.002

to improve the quality of these real-time software systems and thereby reduce the risk to the public of system failure, extensive research has been carried out on formal, mathematical specification, verification and validation techniques. While these methods are beginning to show promise in improving software quality, it is not always clear to software practitioners how well these theoretical techniques model real embedded systems and how they can be applied to practical industrial systems. The jury is still out as to whether some of the formal techniques are really only useful on academic examples. On the other hand, it is also becoming clear to industry and regulators that conventional techniques based on conformance to development processes and on testing techniques, no matter how extensive, cannot guarantee system properties to a sufficiently satisfactory level.

One of the best indicators of this concern is the recently published National Academy of Sciences Report on Software for Dependable Systems [1]. This report, which we predict will prove very influential in guiding regulators and industry, documents the current state of the art in software development - and what needs to be done in the future.

The report emphasizes two *Findings*. The first of these is that software development has to improve in order to deliver more dependable software-based systems in a world in which software plays an ever-increasing role. The second is that we need to document and analyse software failures in order to understand the contributing factors, especially if the contributing factors were related to the development process. In addition, the report makes a number of important *Recommendations*. The recommendations are targeted at two distinct groups: i) Builders and Users of Software, and ii) Supporters of Software Education and Research. Recommendations to the first group include: use formal methods, software development technologies and principles that are known to be effective; build dependability cases that include security concerns; do not rely solely on process and testing to provide dependability; demand transparency and accountability; base certification on inspection and analysis of dependability claims and evidence. Recommendations to the second group include: place greater emphasis on dependability in the education of software professionals/researchers; fund basic research to improve dependability of systems that contain software, with an emphasis on evidence of dependability.

We agree, substantially, with the findings and recommendations of this report. We believe that it is implicit in the report, but should be stated more emphatically, that we need to use what we already know about building highly dependable software, and that we need to conduct more research on how to build *and certify* software-based systems, in which the focus should shift from process to product.

1.1 The Goal of Certification

The goal of certification is to systematically determine, based on the principles of science, engineering and measurement theory, whether an artifact satisfies accepted, well defined and measurable criteria.

1.2 Consortium Objectives

The consortium represents a research endeavour, totally devoid of any explicit endorsement by any regulators/companies. Its aim is to understand certification issues with respect to systems that contain (significant) software components, and to make recommendations on processes and standards that impact on the certification of such systems. We want industry, regulators and universities involved in the consortium so that our recommendations are practical and effective in the real-world. We are interested in certification of systems in medical devices, nuclear power plants, automotive and aerospace. These areas puposely cover a range from the relatively unregulated (Automotive) to heavily regulated (Nuclear). We believe that we need to look at domain specific issues, but also share ideas between the different domains and levels of regulation. If we do a good job of this, then some of our ideas could be taken up in the future by regulator driven open processes and/or standards organizations. To this end, SCC has come up with the following objectives:

(i) To promote the scientific understanding of certification for Systems containing Software (ScS) and the standards on which it is based;

(ii) To promote the cost-effective deployment of product-focused ScS certification standards;

(iii) To promote public, government and industrial understanding of the concept of ScS certification and the acceptance of the need for certification standards for software related products;

(iv) To investigate and integrate formal methods into ScS certification and development;

(v) To co-ordinate software certification initiatives and activities to further objectives i-iv above.

1.3 Goals to Achieve SCC Objectives

Primary Goals

- Develop and document generic certification models that will serve as a framework for the definition of domain specific regulatory and certification requirements.
- Proof of concept: Develop and document software regulatory requirements that help both developers of the software and the regulators of the software in the development of safe, reliable software applications in specific domains.

Detailed Goals

- Use existing software engineering and formal methods knowledge to develop appropriate evidence-based standards and audit points for critical software in specific domains, including hard real-time, safety-critical systems.
- Create software development methods that comply with the above standards and audit points for the development of critical software.
- Research and develop improved methods and tools for the development of crit-

ical software.

2 A Grand Challenge

Surprisingly there is no consistent picture of how the reliability of critical software is currently regulated. Society is starting to demand that software used in critical systems must meet minimum safety, security and reliability standards. Manufacturers of these systems are in the unenviable position of not having clear guidelines as to what may be regarded as acceptable practice. Even where the systems are not mission critical, software producers and their customers are becoming interested in methods for assuring quality that may result in software supplied with guarantees.

It is preferable to consider what standards should be met, and how to assure compliance with those standards before we are forced to act in haste as a consequence of another serious accident or series of software related failures. In this context, SCC, through McMaster University's Software Quality Research Laboratory, has initiated a "Software Grand Challenge" to the software engineering and formal methods communities, involving critical software to manage a real medical device, namely a heart pacemaker [6]. The specification for the device was enthusiastically supplied by Boston Scientific, based very closely on a device they built some time ago. It has all the features of the original device and a microcontroller base hardware reference platform is a crucial component of this challenge.

SCC is also designing an *evidence-based, product-focused certification process* to provide a means of assessing solutions to the challenge. Participants will be encouraged to submit supporting evidence as well as their solutions. This supporting evidence will be used to conduct the certification activities. This will enable the Challenge "community" to explore the concept of licensing evidence (i.e. certification) and the role of standards in the production of such software. Furthermore, it will provide a more objective basis for comparison between putative solutions to the Challenge. The results of the certification activities will be provided to participants and eventually will be included in the publication(s) arising from the challenge.

The major certification body for such devices marketed in North America is the U.S. Food and Drug Administration (FDA). The FDA has already indicated that they will participate actively in this challenge. We are currently trying to get Health Canada to participate as well. The above challenge has been accepted as one of the international "Grand Challenges" promoted by the Grand Challenge Consortium being managed from the UK. We see it as the first of a number of similar challenges, with others in the domains of automobiles and aerospace.

Safety-critical systems can be viewed as the "thin end of the wedge" for the cause of Software Certification because the cost and consequences of system failure currently justify what industry views as the extra expense of Formal Methods. The challenge is to create theories that are accessible to practicing engineers and the tools to make the applications of theory practical for "real-world" problems from industry. As the rigorous techniques become more refined with better tool support,

[6] Details of the Pacemaker Grand Challenge are available at: http://sqrl.mcmaster.ca/pacemaker.htm

the software development industry will see increased application of Formal Methods and acceptance of Software Certification. SCC is therefore initially focusing its certification efforts in the Aerospace, Automotive, Medical and Nuclear fields.

3 Top 9 Hurdles for Software Certification

During the December SCC meeting, participants identified the following 9 hurdles. The first 4 of these were voted as the top 4 hurdles, in the order shown. The remaining 5 hurdles were not prioritized.

(i) *Clarity of expectation and method of communicating with regulator* - Application developers do not know what to produce, and often have to pay consultants to figure it out. In many cases the consultants are not correct. It is also difficult for the regulator to communicate changes in what is expected.

(ii) *Lack of clear definition of evidence and how to evaluate it* - We do not have enough theory that helps us determine the effectiveness of attributes and metrics that indicate dependability in software products. We also have a problem in understanding how to combine different evidentiary artifacts when determining an overall evaluation of the evidence.

(iii) *Poor documentation of requirements and environmental assumptions* - It is trite, but true, that without accurate and complete requirements we have no reliable evidence that the application will provide the required behaviour. It is well know that incomplete/poor requirements and incorrect/unstated environmental assumptions lead to poor applications.

(iv) *Incomplete understanding of the appropriate use of inspection, testing and analysis* - It is by no means clear when we should use inspection, testing and mathematical analysis to achieve specific levels of dependability.

(v) *No overarching theory of coverage that enables coverage to accumulate across multiple verification techniques* - Clearly, no single quality assurance technique is sufficient for effective verification, and as effective evidence for certification. Previous experience shows that a variety of automated formal verification techniques such as static analysis, model checking, and theorem proving as well as conventional testing techniques can each be effective, but each of these differs in strength of properties verified, types of behaviours covered, and the life-cycle stage in which they are most naturally applied. Not only do each of these techniques need to report coverage of behaviours/properties and produce evidence of verification, the coverage/evidence needs to be sharable and cumulative across techniques. Sharing coverage/evidence across techniques via a single unified coverage/evidence framework will (a) enable the successes of one technique to reduce the obligations of accompanying techniques and (b) clarify gaps in verification that must be filled by following techniques. The most convincing arguments of correctness will rely on being able to accurately state in quantitative ways how multiple verification techniques each contribute evidence of overall correctness.

(vi) *Theories of coverage for properties like timing, tolerances as well as programs with concurrency* - Notions of structural coverage for testing play a key role in development and certification of safety-critical software. Though requirements of structural coverage for tests traceable to requirements such as the modified condition/decision coverage (MCDC) mandated in DO-178B are a far-from-perfect mechanism for providing evidence of correctness, they are one of the few notions in existing certification standards that provide a quantifiable assessment of the degree to which a program's behaviour has been examined during the quality assurance process. Unfortunately, existing coverage measures fail to take into account (a) properties such as timing and tolerance ranges for data values and (b) the degree to which interleavings in concurrent computations are exercised. As a result, even development efforts that succeed in achieving high levels of mandated coverage measures often fail to fully explore and validate common sources of program faults.

(vii) *Hard to estimate a priori the V&V and certification costs* - Currently, it is difficult to make a strong business case for introduction of formal techniques or, indeed, for any new techniques, because it is difficult to estimate both (a) the time required to carry out various forms of formal analysis and (b) the reductions that can be obtained either in costs of the certification process itself or long-term costs associated with fewer defects found late in the development life-cycle, greater reuse in subsequent development of similar systems, and fewer recalls of deployed systems, and decreased liability costs.

(viii) *Lack of interoperable tools to manage, reason, and provide traceability* - The result is that small change often requires a large effort. We need tools that scale.

(ix) *Laws, regulation, lawyers and politics* - Certification has legal implications, and there are thus all the normal barriers to straightforward technical decisions. As difficult as the technical problems may be, political considerations complicate the process immeasurably.

4 Conclusion

We believe that rigorous software engineering principles and formal methods should be used to: i) develop dependable software applications that are accompanied by certification evidence of an appropriate level; and ii) develop software certification processes that are focused more on product (including accompanying evidence) than on software development process. The hurdles described above, although arrived at independently, are in substantial agreement with the NAS report [1].

5 SCC Participants

The participants at the December SCC meeting were: Jo Atlee, Rick Chapman, Rance Cleaveland, Darren Cofer, Arie Gurfinkel, John Hatcliff, Mats Heimdahl, Brian Larson, Mark Lawford, Tom Maibaum, Dennis Peters, Oleg Sokolsky, David

Tremaine, Alan Wassyng, Fred Wurden.

References

[1] Jackson, D., J. Bloch, M. Dewalt, R. Gardner, P. Lee, S. B. Lipner, C. Perrow, J. Pincus, J. Rushby, L. Sha, M. Thomas, S. Wallsten and D. Woods, "Software for Dependable Systems: Sufficient Evidence?" National Academies Press, 2007.
URL http://www.nap.edu/catalog.php?record_id=11923