Software Certification Experience in the Canadian Nuclear Industry: Lessons for the Future *

Alan Wassyng
McMaster Centre for Software Certification
McMaster University
1280 Main St W, Hamilton, ON, Canada L8S 4K1
wassyng@mcmaster.ca

Mark Lawford
McMaster Centre for Software Certification
McMaster University
1280 Main St W, Hamilton, ON, Canada L8S 4K1
lawford@mcmaster.ca

Tom Maibaum
McMaster Centre for Software Certification
McMaster University
1280 Main St W, Hamilton, ON, Canada L8S 4K1
tom@maibaum.org

ABSTRACT

The computer controlled shutdown systems for the Nuclear Power Generating Station at Darlington, Canada, have been subject to licensing scrutinization on a number of occasions. After the first licence was approved in 1990, the licensee, Ontario Hydro, was given a number of years by the regulator to redesign the shutdown systems so that they would be more maintainable. This paper briefly describes the original certification process, lessons learned, and the subsequent development and certification of the shutdown systems. The development, internal certification processes and the regulator's certification process are briefly described. Although twenty years has elapsed since this work started, and there are new analysis techniques and tools that could be applied today, the original process itself has withstood the test of time extraordinarily well. This paper describes principles that explain why it was so successful, and how we can develop more modern approaches from this experience.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General—*standards*

General Terms

Documentation, Reliability, Standardization, Verification

Keywords

nuclear, software certification, safety-critical software

1. INTRODUCTION

The computer controlled shutdown systems for the Nuclear Power Generating Station at Darlington, Canada, were the first safety systems in a nuclear power plant in Canada that were implemented in software. After the first licence was granted in 1990, the licensee, Ontario Hydro, was given a number of years by the regulator to redesign the shutdown systems so that they would be more maintainable during the inevitable changes required in the future. This paper briefly describes the original certification process, lessons learned, the subsequent development and certification of the shutdown systems, and some suggestions for improving the processes and tools in light of more recent developments in technology and in regulatory frameworks. We believe that the safety-critical methodology developed over the years, and the certification process adopted by the regulator were largely very successful. This paper describes principles that contributed to the success, and how we can develop more modern approaches from this experience.

*Funded by the Ministry of Research and Innovation through the Ontario Research Fund - Research Excellence (Round 4)

EMSOFT'11, October 9–14, 2011, Taipei, Taiwan.

2. ANCIENT HISTORY

The Darlington Nuclear Generating Station (DNGS) in Ontario, Canada was ready to go online in the late 1980s. The two independent and diverse shutdowns systems, Shutdown System One (SDS1) and Shutdown System Two (SDS2) were the first nuclear safety systems in Canada that were software based. This posed a problem for the regulators in that they had no experience in evaluating the dependability and safety of a computer-controlled safety system. The story of the verification of the original Darlington shutdown systems is the subject of section 2.2. This original verification project is important for a number of reasons:

- We learned fundamental lessons about certification of safety-critical software from this experience, and these lessons formed the basis of a whole new methodology for the development of certifiable safety-critical software-based systems.

- Many researchers think of this original (very costly) experience when drawing conclusions about the Ontario Hydro (OH, but now Ontario Power Generation) approach to safety-critical software development and verification – and that is quite misleading. As we will show (section 3.3), the next generation safety-critical methodology used at Ontario Power Generation (OPG) is much more effective in building and certifying dependable, safe and maintainable software systems.

- The regulator and the licensee had to learn together how to develop and certify safety-critical software so that the licensee did not face an insurmountable, outrageously expensive task, and the regulator was confident that the software was indeed safe, dependable and maintainable, again without facing an insurmountable, outrageously expensive task,.

2.1 Development of the Shutdown Systems for Darlington NGS – 1980s

At the time that the shutdown systems for Darlington were originally created, OH and Atomic Energy Canada Limited (AECL – the manufacturers of the CANDU reactors) already had considerable experience in developing software systems for use in the nuclear industry. They had processes in place and a number of very experienced people. However, as was typical in those days, the primary people involved in this endeavour had a wealth of practical experience, but were not fully aware of some of the principles emerging from academic work in formal methods. It was also a time in which formal methods had a severe image problem, having promised much but delivering much less than promised.

The software was developed (and documented), but the requirements documents were not detailed enough. In spite of the deficiency with respect to requirements, a surprisingly rigorous and comprehensive testing regime had already been developed.

2.2 Verifying the Shutdown Systems for Darlington NGS – 1989 / 1990

Presented with the fully developed and tested shutdown systems, the Atomic Energy Control Board (AECB), which was the Canadian regulatory body at that time, was faced with the task of licensing DNGS, and thus had to evaluate the safety and dependability of the shutdown systems. This was not an easy task, and they turned to Dave Parnas for advice. Parnas' scheme for each of SDS1 and SDS2 was to:

- Have AECL and OH develop a document that was a mathematically precise description of behaviour of the shutdown system. Each document was based on existing requirements and design documents, supplemented by expert knowledge of CANDU shutdown systems. Also, each document was at a level somewhere between software requirements and software design, and the behaviour was documented using tabular expressions.

- A team of analysts was given the task of developing tabular expressions, called *program function tables* (PFTs) for each subprogram in the code. The PFTs thus represented behaviour of each subprogram in a form that related inputs to the program to the outputs of the program. This presents a very different view of the code, since intermediate variables, so prevalent in code, are all replaced by equivalent expressions involving only inputs and outputs to the program (and, perhaps, their 'previous' values).

- Another team of analysts (completely independent from the developers of the PFTs) then performed a comparison of the tabular expressions in the reference document with the PFTs developed from the code. Mathematical differences between these tabular expressions then had to be further analyzed to ascertain whether or not the differences resulted in differences in behaviour.

- After the analysis was completed, a walkthrough was conducted at which representatives from AECB and OH/AECL met so that the results could be presented to the AECB. The walkthrough was moderated by an independent consultant to the AECB.

Two (slightly different) views of this original verification were published: one by people associated with the regulator [16]; and the other by members of the team that performed the analyses [1].

2.3 The Outcome

The eventual outcome of this verification activity proved to be extremely interesting. After completing this very demanding effort – demanding for both the regulator and the licensee, there were a number of important outcomes.

- DNGS was provisionally and conditionally licensed for a specified number of years. At the time, the regulator was satisfied that the shutdown systems would operate correctly and safely.

- The license to operate was an important outcome since the plant was ready to go online, and the delay in obtaining regulatory approval proved to be extremely costly.

- OH was informed that they would have to redesign the shutdown system software so that it was not only safe, but that it would be maintainable. The initial license allowed no changes to the shutdown systems because the designs were judged to be not maintainable enough, i.e., the regulator felt that if changes were made to SDS1 or SDS2, it would be difficult to evaluate the safety of those changes without repeating the entire verification process described in the previous section – plus testing, of course.

- OH decided that, if they were to continue to use software for safety-critical applications, they needed to learn from this extremely costly lesson, and develop a rigorous approach to building and verifying such systems. The safety culture at OH and now OPG is extremely strong, and the way in which they tackled this problem is a lesson for all companies/organizations involved in the development of safety-critical systems.

- Verification after-the-fact, i.e., post-development, is much more difficult than it needs to be.

- The verification step, from code back to (almost) requirements level is large and complex, and there are numerous good reasons why it would be better to verify a series of interim results.

3. MORE MODERN HISTORY

This section describes the development of OPG's current approach to the development of safety-critical software. Details of the technical aspects of this methodology have been published in [17, 13, 19, 14]. Most aspects of this approach are also used at AECL. This methodology was developed very carefully so that it would have a number of specific attributes:

- Software developed using this methodology will be *'correct'* with respect to its requirements; *maintainable*, in that foreseen categories of changes can be made so that the changes are predictably isolated; and that the mathematically precise requirements can be read and understood, but not necessarily developed by nuclear engineers.

- The software will be *dependable* even in the event of hardware failures, in which case it will be 'very likely' to result in *safe* behaviour even if it cannot evaluate what would constitute *correct* behaviour at that moment.

- The quality and correctness of the software can be evaluated relatively easily by a regulator.

3.1 Shutdown System Studies

Knowing that redesigned versions of SDS1 and SDS2 would be required within a short number of years, OH embarked on a series of research projects dubbed the *shutdown system studies*. These studies were components of the *"CANDU Shutdown System Study – Computerized Trip Logic"*, which examined a variety of topics that arose during the DNGS shutdown systems' developments, verifications and walkthroughs. Typical topics were programming languages for safety-critical systems, hardware replacements (the hardware at that stage was already old), self-checks and fault tolerance, system requirement specification and verification, enforced design diversity, guided inspections, software organization, and others.

These studies laid the initial basis for the eventual OH/AECL methodology for building safety-critical software applications as described in the remainder of section 3.

3.2 The CANDU High-Level Standard

The CANDU Standard for Safety-Critical Software [12] describes the software development and verification principles and attributes required. The standard is supplemented by a collection of lower-level standards that provide more detail for many of the procedures required.

The standard made use of existing international standards of the day to ensure that it was in line with process standards used throughout the nuclear industry, but was somewhat different from such standards by placing slightly more emphasis on the delivered product rather than simply mandating attributes of the process.

3.3 The Redesign Project

One of the primary goals of the *redesign project* was to use *information hiding* to achieve a design that would be extremely robust with regard to (predicted) future changes, thus satisfying the regulator's requirement that the system had to be more maintainable than was the original version. At the same time, OH and AECL used this opportunity to research and document a comprehensive and *integrated* methodology (see [19]) for the development of safety-critical software. The project life-cycle was based on a spiral model implementation of the major steps shown in figure 1. The spiral model still imposed hard pre-requisites for starting an activity. The figure also shows the tools that were used in different phases of development and verification.

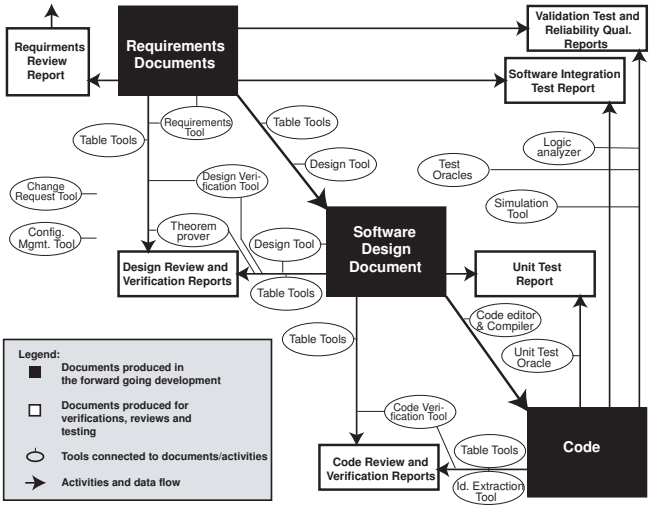

Figure 1: Life-cycle documents and tools. [18]

3.3.1 Lower-Level Standards

Lower-level standards (called *procedures* at OPG) were researched and developed for all the activities in the life-cycle. Some of the procedures applied to both SDS1 and SDS2, but in many cases, there were different versions for SDS1 and SDS2. This occurred mainly because of the diversity enforced over the two systems. For SDS1, for example, there are procedures for *Hazards Analyses*, *Trip Computer Design Requirements*, *Trip Computer Design Description*, *Requirements Review*, *Software Design Description*, *Software Design Review*, *Software Design Verification*, *Coding*, *Code Review*, *Code Verification*, a variety of testing procedures, and a procedure for maintenance and revision of all documents. All of these are specified in a *Software Development Plan*. Most of these "Procedures" typically have two major sections. The first is the 'process'. It includes how activities should be performed – and why they should be performed – and why they should be performed in that manner. The *why* is different from typical standards, and was the subject of significant criticism at the time. There was a definite preference amongst many software engineers that standards should restrict themselves to a simple statement of what should be done. Current thinking seems to vindicate our original approach. *Rationale* is as important in the documentation of standards as it is in the documentation of software artifacts. The second major section is one on 'documentation'. The documentation requirements serve to clarify the expected outcomes from the process, and also serve as a template. This helps both developers and certifiers. The actual process and lessons learned were described in [17]. Note that the regulators were kept in the loop during the production of these lower-level standards – and this included Dave Parnas, who acted as a consultant throughout those years.

3.3.2 Development

The major steps in the software development (including verification activities) were briefly described in [17, 18]. The following points are worth highlighting:

- The requirements are (mainly) documented in a mathematical format, primarily through the use of tabular expressions. The mathematical basis enables a precision not (easily) possible using natural language. The *function tables* are relatively easy for nuclear engineers to read and understand so that they can evaluate the specified behaviour.

- The software design is also documented (mainly) through the use of *function tables*. Since both the requirements and design are described mathematically it is possible to conduct a mathematical verification of the design against requirements.

- Assuming that the programming language has a well-defined semantics, and that the semantics is preserved by the compiler, code is always a mathematical object, so a mathematical verification can also be performed against the design.

- We have thus satisfied one of our major objectives, derived from lessons learned during the original verification (section 2.3), namely that the verification should be performed in at least two steps, each significantly smaller than the original one was.

- The principle of *information hiding* was substantively integrated into the development process. Lists of *likely changes*, and, a natural adjunct, lists of *unlikely changes* are developed during requirements and early stages of the design, and are used to encapsulate each likely change in a single software module. Constants are defined in ranges rather than as distinct values. The ranges are used during mathematical verifications to ensure that changes to constants can be made without incurring new verifications – as long as the changed values remain within the predefined ranges.

- *Hardware hiding* modules are used to isolate the software from changes in hardware, to make it possible to use the requirements to describe directly the behaviour of software modules, and also to facilitate the software design verification.

- *Supplementary functions* are used to document design decisions, module interfaces, and to facilitate software design verification, by making it possible to perform the verification piece-wise.

- Techniques to facilitate verification were introduced into the forward-going process. For example, each software subprogram is designed according to a *get-process-set* protocol. This means that all inputs are fetched at the start, followed by all required processing, followed by output. If this is not done, mathematical verification can be much more difficult.

- *Fault tolerance principles* were used to ensure that safe behaviour was significantly more likely than unsafe behaviour in the face of hardware failure.

3.3.3 Certification

The regulators were aware of the process developed at OH, and were, in principle, in agreement with the process. Also, the process defined documents (products) that had to be delivered, as well as the attributes that had to be observable within the documents. Since both the development team and the regulator believed that 'correctness' was the most important attribute, the requirements and the verifications steps assumed greater importance than other aspects of the development output with respect to certification. The regulators 'audited' the results by selecting a slice and then conducting a walkthrough of the slice. We think this makes excellent sense as a model for certification. It adds significantly to reasons why the regulator should be concerned with *people* and *process*, not just *product*. Given that the people and process pass inspection, the regulators do not need to examine every aspect of the software product. It should be sufficient to check the results for a representative 'slice'. Since the developers cannot predict the slice ahead of time, confirmation of attributes and 'correctness' of the slice should be sufficient to gauge the dependability and correctness of the software system. We believe this kind of approach will work well even if we use *assurance/safety cases* [7, 3] to present the 'argument' to the regulators.

4. CERTIFICATION LESSONS

While re-evaluating the DNGS experience that took place during the period 1989 through 2002, a number of lessons regarding software certification have become clear.

- Agreement on the certification process between regulator and licensee is incredibly useful in making the certification process *predictable*. In the case under discussion, this was possible because the number of licensees the nuclear regulator had to deal with at the time was small, so that adequate time could be spent in understanding the respective positions. If this is not the case, which would be typical in many jurisdictions, we could rephrase this lesson to say that detailed understanding of a precise set of requirements of the certification process will make that process predictable – a huge plus for both the regulator and the licensee.

- The *certification tripod* of people, process, product is important in building the confidence of the regulator, but the regulator does not need to treat the three components on an equal footing. For instance, the regulator may be satisfied to get third party certification for the people and process components, and then conduct an audit of the product. There are many variations on this theme, and different jurisdictions will have different legal needs, capabilities and approaches.

- To emphasize the previous point: given confidence about the people and process used in developing (and verifying) a product, one excellent way for a regulator to examine the software product is to conduct an audit of a slice through that product. The chosen slice would be specified only after completion of the development process.

- The licensee must have confidence in the capabilities of the regulator. There will inevitably be some contentious issues in the certification of a safety-critical product. The licensee and regulator must be able to discuss the issues on an equal footing. The important points here are that the regulator and the applicant both have professional reputations, and that the

regime imposed by the regulator is detailed enough and unambiguous enough so that there are no arguments about interpretation.

- There is a truism in most engineering projects – quality cannot be achieved by testing/verification after the fact. Quality has to be built into the development process from the start, and this includes a number of important items:
  - Rationale should be explicitly stated in the requirements, design, and implementation – and also in the appropriate standards and certification processes.
  - The forward going development process should be designed to aid verification techniques.
  - The forward going development process should be designed to aid the construction of the safety case in conformance with the regulator's requirements.
  - Serious consideration and effort must be expended in constructing a *defense-in-depth* approach, and the various constituents of this approach should be made explicit to all concerned. This holds both for the issues related to safety of the system, as well as the evidence used in the safety argument. (For example, a proof of correctness using a theorem prover that is not itself verified to be correct needs to be supplemented with further evidence to provide confidence in the result, e.g., another proof using a different theorem prover.) This includes documenting it for the certification authority.
- Mathematical precision can help build the confidence of both the development team and the certification authority. However, we need to guard against misplaced confidence as well. We make mistakes in mathematics just as we do in other disciplines. We need to recognize that we are not (yet?) capable of describing everything mathematically and have it readable, understandable, and amenable to analysis. However, we should also realize that if our requirements are not stated mathematically, we will be unable to perform mathematical verifications of design and implementation. These mathematical verifications complement testing and are an essential element in our defense-in-depth strategy to build confidence in the safety/correctness of our product.
- Arguments related to completeness are also essential to building confidence in our system. Aspects of the process that enable us to check for gaps in 'coverage' are extremely beneficial. We are used to this concept with respect to testing, but it applies to many other facets as well. One of the reasons we recommend the use of tabular expressions is their capability to ensure that we have covered the whole input domain of the function.

5. OUR CURRENT VIEW

The OPG/AECL safety-critical software methodology has stood the test of time. We believe it has helped teams produce highly-dependable software, and has provided a stable, comprehensive and understandable platform to serve as a basis for the regulator (now called the *Canadian Nuclear Safety Commission*). We believe that this set of standards defines an implicit assurance/safety case, and that we will be able to effectively package in a safety case the products produced by following that standard effectively in a safety case [20]. Some brief thoughts related to this idea are presented in the remainder of section 5.

5.1 Advances in Theory and Tools

Most of the research related to the development of the methodology was conducted in the early 1990s. A number of crucial advances have been made in the meantime. One of the most important is the advent of *model checking* [5]. *Simulation of tabular expressions* was in its infancy [9]. Both of these would enable us to analyze our requirements in ways that were just not possible at that time. Proof-checkers could be used to provide confirmation that proofs produced by theorem provers (we used PVS [15]) could be used without being backed-up by manual proofs. The problem with tool support in general is that most certification authorities in safety-critical domains do not allow tools to be used where they are the sole means of performing a task, if the tool is not qualified to the same level of criticality as the application it is used on.

Model checkers have improved their applicability to software [11] and enhancements, such as symbolic model checking [4], have allowed the technique to scale to industrial sized problems. SAT and SMT solvers are now being integrated with theorem proving tools [8]. The SCR* toolset now not only has advance simulation capabilities but also integrated analysis techniques such as model checking [10]. Recently we have created a prototype tabular expression toolbox for model based engineering with tabular expressions from within Matlab/SIMULINK [6]. The toolbox allows for verification of the correctness of the tables by integrating PVS and the CVC3 SMT solver [2] to provide redundant checks of some properties.

5.2 Advances in the Development Process

The software development process has not changed much in the time elapsed since 2002. The OPG methodology already incorporated many of the features that are now growing in importance, for example, rationale and traceability were already key features. More modern programming languages than the ones used for DNGS (Pascal and FORTRAN 66) would make life easier, but the coding is really a small part of the effort. Tool support for methodologies has improved, but we did not mention such tools in section 5.1, for the simple reason that commercial tools do not adequately support the OPG process. The tools that were mentioned in that section would have a definite impact on the development process and their functionality should clearly be introduced into the methodology. We strongly believe that the development of effective tools will improve substantially when the idea of integrating the development methods (see [19]) becomes more commonplace. Truly integrated toolsets can be developed from integrated methods.

5.3 Advances in the Certification Process

For the longest time there was very little new in the certification process suggested by nuclear regulators in North America. Recently, there has been considerable activity in

the U.S. Nuclear Regulatory Commission (US NRC), as they build their capability in certifying digital instrumentation and control. A great deal of interest has been voiced in the use of assurance/safety cases. We believe that safety cases will actually represent a significant step forward in certification of safety-critical systems in North America – they are widely used in the U.K., and in Europe. However, we caution that there is significant work to be done still before safety cases are ready for day-to-day documentation of licensing submissions. Major problems that we feel need to be addressed are that there is little to no theory that tells us how to:

- Make trade-offs between different combinations of evidence.
- Explicitly map claims to supporting evidence in a consistent and thus predictable way.
- Evaluate the supporting evidence for claims through explicit justification provided in the assurance case.
- Identify gaps in the completeness arguments presented.
- Present assurance cases within a domain in a consistent way so that certifiers can build expertise in reviewing assurance cases.

6. CONCLUSIONS

Today's regulatory regimes in domains like nuclear power and medical devices are undergoing redevelopment in light of pressures that are both technological, administrative and social. Considering the last first, there is growing awareness, because of many news stories reporting disasters, among people around the world of both the prevalence of unsafe software and systems in the world and the apparent deficiencies of regulatory regimes. In order to stop the "bad press", regulators are having to examine what causes the negative impact, both in technological terms and in terms of the effectiveness of their regulatory regimes. Certainly, technological developments have increased support for the scenarios described above. Model checking and constraint solving tools have made analysis of software systems significantly more effective. This aids the manufacturer significantly in supplying evidence to support safety and efficacy claims. The US FDA has recently requested that applicants for licensing insulin infusion pumps should submit a safety case as part of their application. This is to help manufacturers to identify hazards more effectively and to aid the FDA in assessing the application. In the nuclear domain, manufacturers are attempting to use FPGA technology to both replace older analog subsystems, as well as in new builds to implement some aspects of safety and control systems. There are pressures in the nuclear domain to relax requirements for separation of safety and control functions, as well as to relax requirements on physical and logical separation of certain subsystems. These pressures act to increase the complexity of safety subsystems, making the safety claim harder to support and certainly making life much harder for the regulators. It would appear that recent technological developments both aid and hinder the discussion of safety in software based systems.

The lessons learned from the OH/OPG experience are salutary in light of these developments and pressures. Certainly, of primary importance is the insistence on separating safety and control in safety-critical systems. Integrating safety and control just makes the resulting system much too complex to provide guarantees of safety. If one day we have complete mastery over software analysis for software of any size, we might wish to abandon this design heuristic. With the technology of today, and for the foreseeable future, the public and regulators cannot and should not support such moves. An interesting aspect of this experience noted above was that the case made for safety had the substance, if not the form, of what is today called the safety case. The requirement in safety cases of having well defined (testable in the scientific sense) claims, supported by evidence about the product and connected by arguments about how claims are supported by the adduced evidence, were present in the regimes developed at OH/OPG with the active participation of the regulator. The regulator's requirement for the separation of safety and control was of immeasurable importance in making the effective guarantees of safety possible.

7. ACKNOWLEDGMENTS

The work presented in this paper represents the combined efforts of many current and former employees of Ontario Power Generation and AECL, including: Glenn Archinoff, Dominic Chan, Rick Hohendorf, Roland Huget, Paul Joannou, Peter Froebel, David Lau, Elder Matias, Jeff McDougall, Greg Moum, Dave Vermey, Mike Viola, and Alanna Wong. A special acknowledgement is due to David Parnas. This work represents the successful application of many of his ideas regarding software engineering, and his guidance and mentorship during those years was truly invaluable.

8. REFERENCES

[1] G. H. Archinoff, R. J. Hohendorf, A. Wassyng, B. Quigley, and M. R. Borsch. Verification of the shutdown system software at the Darlington nuclear generating station. In *International Conference on Control and Instrumentation in Nuclear Installations*, Glasgow, UK, May 1990. The Institution of Nuclear Engineers.

[2] C. Barrett and C. Tinelli. CVC3. In W. Damm and H. Hermanns, editors, *Proceedings of the* 19^{th} *International Conference on Computer Aided Verification (CAV '07)*, volume 4590 of *Lecture Notes in Computer Science*, pages 298–302. Springer-Verlag, July 2007. Berlin, Germany.

[3] P. Bishop and R. Bloomfield. A methodology for safety case development. In F. Redmill and T. Anderson, editors, *Industrial Perspectives of Safety-critical Systems: Proceedings of the Sixth Safety-critical Systems Symposium*, pages 194–203, Birmingham, UK, 1998. Springer.

[4] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and computation*, 98(2):142–170, 1992.

[5] E. Clarke. Model checking. In *Foundations of software technology and theoretical computer science*, pages 54–56. Springer, 1997.

[6] C. Eles and M. Lawford. A tabular expression toolbox for matlab/simulink. In *3rd NASA Formal Methods*

Symposium, volume 6617 of *LNCS*, pages 494–499. Springer-Verlag, 2010.

[7] E. Fong, M. Kass, T. Rhodes, and F. Boland. Structured assurance case methodology for assessing software trustworthiness. In *Secure Software Integration and Reliability Improvement Companion (SSIRI-C), 2010 Fourth International Conference on*, pages 32–33. IEEE, 2010.

[8] Formal Methods Program. Formal methods roadmap: PVS, ICS, and SAL. Technical Report SRI-CSL-03-05, Computer Science Laboratory, SRI International, Menlo Park, CA, Oct. 2003.

[9] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. SCR*: A toolset for specifying and analyzing requirements. In *Compass '95: 10th Annual Conference on Computer Assurance*, pages 109–122, Gaithersburg, Maryland, 1995. National Institute of Standards and Technology.

[10] C. Heitmeyer, J. Kirby, B. Labaw, and R. Bharadwaj. SCR*: A toolset for specifying and analyzing software requirements. In *Proc. 10th Int. Conf. Computer Aided Verification (CAV'98), Vancouver, BC, Canada, June-July 1998*, volume 1427 of *Lecture Notes in Computer Science*, pages 526–531. Springer, 1998.

[11] G. Holzmann. Software model checking. volume 180, pages 309–355. IOS Press, Computer and System Sciences, Marktoberdorf, Germany, Aug. 2000.

[12] P. Joannou et al. Standard for Software Engineering of Safety Critical Software. CANDU Computer Systems Engineering Centre of Excellence Standard CE-1001-STD Rev. 1, Jan. 1995.

[13] M. Lawford, J. McDougall, P. Froebel, and G. Moum. Practical application of functional and relational methods for the specification and verification of safety critical software. In T. Rus, editor, *Proceedings Algebraic Methodology and Software Technology, 8th International Conference, AMAST 2000, Iowa City, Iowa, USA, May 2000*, volume 1816 of *Lecture Notes in Computer Science*, pages 73–88. Springer, 2000.

[14] M. Lawford and A. Wassyng. Formal verification of nuclear systems: Past, present and future. In V. Kharchenko and T. Tagarev, editors, 1^{st} *International Workshop on Critical Infrastructure Safety and Security (CrISS-DESSERT'11)*, volume 1, pages 43–51, Kirovograd, Ukraine, 2011. National Aerospace University, Kharkiv, Ukraine.

[15] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, Feb. 1995.

[16] D. L. Parnas, G. J. K. Asmis, and J. Madey. Assessment of safety-critical software in nuclear power plants. *Nuclear Safety*, 32(2):189–198, Apr.–June 1991.

[17] A. Wassyng and M. Lawford. Lessons learned from a successful implementation of formal methods in an industrial project. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME 2003: International Symposium of Formal Methods Europe Proceedings*, volume 2805 of *Lecture Notes in Computer Science*, pages 133–153, Pisa, Italy, Aug. 2003. Springer-Verlag.

[18] A. Wassyng and M. Lawford. Software tools for safety-critical software development. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(4–5):337–354, Aug. 2006.

[19] A. Wassyng and M. Lawford. Integrated software methodologies - An engineering approach. *Transactions of the Royal Society of South Africa*, 65(2):125–136, Oct. 2010.

[20] A. Wassyng, T. Maibaum, M. Lawford, and H. Bherer. Software certification: Is there a case against safety cases? In R. Calinescu and E. Jackson, editors, *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, volume 6662 of *Lecture Notes in Computer Science*, pages 206–227. Springer Berlin / Heidelberg, 2011.