

Hazard-driven Testing of Safety-Related Software

J. Joyce, Ph.D., K. Wong, M.Sc.; University of British Columbia; Vancouver, Canada

Keywords: safety, hazard, software, testing, verification

Abstract

This paper argues that the “safety verification” of a safety-related software system needs to be distinguished from the task of verifying that the behaviour of the system conforms to the requirements. Limitations of requirements-based testing are discussed. The main characteristics of a hazard-driven approach to safety testing of software-intensive systems are outlined. This paper also briefly describes an iterative, exploratory approach to safety verification that aims to expose operationally realistic conditions under which unsafe behaviour may occur.

Introduction

Many software professionals assume that the primary difference between developing safety-related software and other kinds of software is simply a matter of “more testing”. But this does not necessarily entail any fundamental difference between the way that ordinary requirements are “proved” during acceptance testing and the approach taken with the verification of safety-related requirements. In this paper, we argue that effective safety testing of a software-intensive system needs to be approached in a manner that differs fundamentally from requirements-based testing.

The practice of software safety has advanced to the point where it is common to see safety requirements derived for a safety-related system. In some cases, the purpose of such requirements is to reduce the likelihood of a hazard occurrence by imposing constraints on the behaviour of the software under certain conditions. Other safety requirements specify the behaviour of protection functions intended to reduce the severity of the consequences due to an occurrence of a hazard. Verifying that the safety requirements are satisfied as part of acceptance testing will provide some evidence that the overall safety risk has been mitigated.

But for a variety of reasons discussed in this paper, the specification and verification of safety requirements falls far short of what is needed to achieve adequate mitigation of safety risk. Unfortunately, the safety analysis of a software-intensive system is sometimes conducted as a detached, theoretical, somewhat passive exercise far away from the mainstream development effort. When this happens, safety engineers may do little more than propose some safety requirements during project inception and then re-appear months or years later to confirm that the safety requirements have been verified. Such an approach is not likely to achieve adequate mitigation of safety risk.

A highly effective way to bring safety analysis into the mainstream of development is to extend the safety analysis task to include hazard-driven verification testing. To be effective, safety testing cannot simply be “more of the same”. It is not re-verification of the safety requirements. Instead, this effort must be driven by insights about the safety behaviour revealed by the analysis of hazards (which usually happens after the safety requirements have been established). Moreover, the goal of safety testing is not to demonstrate the absence of hazards (e.g., that the safety requirements are satisfied) but to search for evidence of their existence. Mindful of Dijkstra’s observation that “Program testing can be used to show the presence of bugs, but never their absence”, we can still

develop some confidence that the risk associated with a particular hazard is acceptable when an aggressive, exploratory attempt to force the occurrence of a hazard under operationally realistic conditions is not successful.

A portion of the safety verification effort may be spent on “filling holes” in the volume of test evidence being assembled by safety engineers in support of their conclusions about the residual safety risk associated with the delivered system. Such holes typically correspond to certain safety related behaviours that are not explicitly addressed by the requirements. The number and size of these holes will likely be fewer and smaller in the case of a system with a very extensive set of safety requirements than for a system with a less extensive set of safety requirements. But even for a system with a very extensive set of safety requirements, there will likely be a need for additional test evidence motivated by details in the hazard analyses. For example, such details may involve particular scenarios or particular interactions between autonomous behaviours that are simply too design-specific or singular to be captured in the form of safety requirements. In short, we cannot rely on the specification of safety requirements to exhaustively communicate what test evidence is needed by the safety engineers to support their conclusions about the residual safety risk associated with the delivered system.

Regrettably, standards and guidelines frequently used to plan, organize and execute a safety management plan for a software system do not necessarily encourage a hazard-driven approach to safety verification. Most of these standards and guidelines refer to the need to review the results of testing safety requirements. Some of these standards and guidelines also refer to the possibility of using specialized verification techniques such as formal methods. One particular guideline, DO178B (ref. 2) is more specific about the verification of safety-related software systems than most other standards and guidelines of this kind. In addition to requirements-based testing, DO178B provides specific details about techniques for achieving extensive test coverage based on code structure. Although such techniques are likely to contribute to the overall quality of the software, they only address generic causes of failure (e.g., “incorrect interrupt handling”, “failure to satisfy execution time requirements”, “stack overflow”) and can be performed largely without any awareness of the results of hazard analysis or even an awareness of what hazards have been identified. We are not aware of any standard or guideline that recognizes limitations of requirements-based testing for the purpose of mitigating safety risk or that provides specific guidance on a hazard-driven approach to safety verification. The lack of such guidance has motivated this paper.

For program managers and other stakeholders with scheduling and budgetary responsibilities in the development of safety-related software, this paper explains why it is necessary to make provisions for a hazard-driven approach to safety testing beyond routine verification of requirements. Should their software be implicated someday in an accident, a moral or legal defense based merely on the claim that every safety requirement was verified may be inadequate. Such an argument may be refuted by evidence that the hazard analyses contains more detailed safety information that could have guided test engineers towards finding initiators of specific hazards.

For test engineers who have been tasked to support safety testing, this paper explains why they should not simply expect to be given a list of requirements to be verified. Instead, they must be prepared to combine their “hands on” knowledge of the actual behaviour of the system with insights gleaned from the hazard analyses in an attempt to expose potential hazard causes. This paper also briefly describes how test engineers may use information typically found in the analysis

of a hazard in an attempt to find a combination of external and internal conditions that may initiate an occurrence of the hazard.

Requirements-based Testing

All rigorous approaches to software development involve some form of requirements specification as well as a phase of system level testing intended to demonstrate conformance of the implemented behaviour with the requirements. The requirements are generally specified at an early stage of system development. Towards the end of development, these requirements are then verified by a set of test cases selected to demonstrate the behaviour expressed the requirements.

At the level of acceptance testing, the verification of a requirement is typically best described as a demonstration of the behaviour specified by the requirement. As a demonstration of the required behaviour, testing is likely to be minimal and straightforward rather than comprehensive and probing. For example, a requirement of the form “The system shall accept input values in the range 28.0 and 32.0.” may be verified in an acceptance test by a single test case with an input value within this range, e.g., 30.0. Lower levels of testing such as unit testing would typically subject the software to a greater variety of test inputs, e.g., 27.9, 28.0, 28.1, 31.9, 32.0 and 32.1. This is partly a consequence of fact that acceptance testing is typically more formal, and hence, more expensive than lower levels of testing.

It is also important to consider the fact that requirements-based testing is typically performed with a predilection for demonstrating that the behaviour of the system conforms to the requirements. Progress is often measured in terms of “requirements successfully verified” and performance incentives may even be linked to achieving a scheduled milestone based on conformance. A test engineer who tries “to break the system” especially when this jeopardizes milestones risks the friendship and support of his or her fellow team members as well as opportunities for career advancement. We do not suggest that test engineers will knowingly overlook genuine problems. We are merely observing that the task of requirements-based testing is typically performed under a set of conditions that encourages producing simple and straightforward test cases to demonstrate conformance.

Role of Testing in Software Safety

The safety analysis of software-related systems has the following main purposes:

1. identify hazards and analyze their causal factors in concert with hardware and human elements
2. mitigate hazards based on their causal factors to reduce the associated safety risk to acceptable levels subject to resource limitations
3. document the residual safety risk for the delivered system including recommendations for additional mitigations.

Testing can contribute to all of these goals, especially when a portion of the overall test effort is driven by an understanding of these hazards and related safety concerns. During development, test results that reveal potentially unsafe behaviours can motivate changes to these behaviours to reduce the associated safety risk. Test results are also an important source of “evidence” for the

conclusions made in the final system safety report about residual safety risk for the delivered system.

During development, testing may reveal a problem in which the behaviour of the system is both unsafe and non-conformant, i.e., violates the requirements. In such a case, the behaviour is likely to be corrected to achieve conformance. A more interesting situation is when the behaviour is “not non-conformant” but is suspected of being hazardous. (The term “not non-conformant” is an euphemism used to describe a behaviour that may be unexpected and perhaps even undesired but, strictly speaking, does not conflict with any particular requirement.) This situation may arise when the requirements are ambiguous, silent or simply wrong. It may be difficult to persuade stakeholders to support the cost of changing the system behaviour when a requirement has not been violated. However, concrete evidence in the form of a test result along with an analysis that the behaviour can lead to a mishap is likely to be more compelling than a hypothetical scenario. Thus, testing may benefit safety objectives during development by serving as a catalyst for change to the system particularly in situations where the requirements have not been violated.

At the end of development, the safety engineering process should yield a final system safety report to document conclusions about the safety of the delivered system. Relevant test evidence should be cited in this report when making conclusions about the safety of the system. A key component of this report is an assessment of the residual risks and a list of recommendations for additional mitigations. We generally expect these conclusions to be positive, e.g., the system may be used safely under certain conditions. However, this report might also include less positive conclusions that identify limitations or outstanding problems that constitute safety risk. These conclusions must be clearly communicated to stakeholders who will ultimately decide whether to deploy the delivered system and the conditions under which it may be deployed. The provision of test evidence to support “unpleasant” conclusions in the final system safety report will be more difficult to overlook than hypothetical scenarios. Thus, testing may benefit safety objectives at the end of development by amplifying and substantiating conclusions about residual safety risk.

There are important differences between testing a software system for the purpose of assessing safety risk and testing electrical, mechanical, chemical and other kinds of non-software systems. In general, the safety testing of a non-software system is based on an assumption that most mishap sequences are initiated by failures. Therefore, such systems are tested under conditions that produce or simulate external conditions that may induce failures, e.g., extreme temperatures, extreme vibration. However, unsafe behaviors by software systems are not always the result of failures. Unsafe behaviours may originate from the requirements due to ambiguities or omissions. It is even possible the requirements are unsafe – but not recognized to be unsafe. Therefore, reliance on testing that involves subjecting the system to extreme conditions (i.e., “shake and bake” testing) is not likely to be as effective in the case of software-intensive systems as it may be for non-software systems. This means that generic strategies for software testing such as boundary condition testing may not be effective in finding unsafe behaviours. Instead, effectively testing a software system for unsafe behaviours generally depends on understanding hazards and the unsafe conditions that could result in the occurrence of a hazard.

Limitations of Requirements-based Testing for Safety

A common approach to safety verification by developers and testers is simply to regard this task as a subset of the overall requirement verification task in which the actual behaviour of the system is compared to the expected behaviour as specified by the requirements. This subset corresponds to a

subset of the requirements that have been identified as “safety requirements”. It is possible that extra care may be taken in developing test cases for the safety requirements and reviewing the test results. However, this approach tightly couples the safety verification effort with the overall process for verifying that the behaviour of the delivered system conforms to the specified requirements.

Such an approach is appealing to project planners and managers because it simplifies the task of estimating the cost of safety verification by reducing this calculation mostly to an estimate of the number of the safety requirements. Similarly, this approach allows progress to be easily tracked using the mechanisms established to track progress of the overall requirements verification process. And perhaps most appealing of all, the completion criteria is rigid leaving no uncertainty about when the safety verification task has been completed. Taken to an extreme, this approach may reduce the safety engineering process to a matter of specifying safety requirements and reviewing the results of requirements based testing.

Safety verification of a software-intensive system in this manner is woefully inadequate for many reasons including:

1. As already mentioned, the task of requirements-based testing is typically performed under a set of conditions that encourages producing simple and straightforward test cases to demonstrate conformance. These test cases are very likely to be based on the same “normal path” logic used by the developer. They are easier and less problematic to develop than a test case that involves more complex combination of conditions which may not have been anticipated by the developer.
2. Test engineers frequently develop an intuitive sense of the “weak points” in the behaviour of a system. Given sufficient latitude, they may be able to find evidence of unsafe behaviour using this intuition, e.g., “I don’t know what I’m looking for, but I’ll know it when I see it.” But the schedule and budgetary constraints of requirements-based testing strongly discourage exploratory testing.
3. Even if the delivered software is defect-free (i.e., conforms 100% to the requirements), its behaviour may be unsafe. The requirements could be ambiguous, incomplete or simply wrong. For a non-trivial software system, there is little hope of completely specifying every detail of a behaviour that may be safety-relevant.
4. With the increasingly integrated nature of software systems, it is unrealistic to assume that the requirements of a safety-related system can be neatly divided into a set of safety requirements and “other” requirements. Some requirements will be obviously relevant to safety while others will be considered unlikely to be relevant. However, there will likely be a significant portion of requirements in between these two extremes whose safety relevance may be uncertain.
5. The output of requirements-based testing is usually binary: “pass” or “fail”. The tools and processes in place to support the overall verification effort may not be amenable to a third possibility “correct, but unsafe” – that is, when the behaviour is “not non-conformant” but is believed to be unsafe. Such results may be ignored or lost as a footnote buried deep in a report on the outcome of verification.

6. Relying exclusively on requirements verification as a source of test evidence for conclusions about the safety of the system is likely to preclude consideration of new hazards, hazard causes or miscellaneous safety concerns that arise during the course of the development after the requirements have established. Even though very few software systems are delivered without changes to the requirements in the course of development, it can be very difficult to persuade stakeholders to accept such changes. The amount of effort to achieve this change may far exceed the amount of effort required to extend the test effort to cover a new concern not explicitly addressed by the existing requirements. In the end, the desire to use testing to explore a particular safety concern may be caught between two hard positions: "Insufficient reason to add more requirements" and "Without a new requirement, no additional testing".

Requirements-based testing may provide some useful results for safety verification. In particular, failed test cases should be examined for possible safety implication by engineers familiar with the identified hazards. But even more importantly, attention should be given to safety-related observations about behaviours deemed to be conformant. Whereas a rigorous development process will address failed test cases, “safe but not non-conformant” behaviours may easily fall through the cracks.

An Approach to Safety-Related Testing

Rather than relying on requirements-based testing, we suggest that the development of a safety-related, software-intensive system must include a separate, hazard-driven test effort to reduce safety risk. A separate, hazard-driven test effort will also allow more complete test evidence to be obtained in support of conclusions about the safety of the delivered system. Key characteristics of this approach to safety verification include:

1. The analysis of the hazards should be used to define the scope and depth of safety verification.
2. The potential of a behaviour to contribute to the occurrence of a hazard rather than conformance of the behaviour to the requirements should be used to determine which results warrant follow-up action.
3. Although the safety verification effort should ultimately produce a documented set of test procedures, a portion of the safety verification effort should be exploratory.
4. Management should avoid measurements of progress for safety verification that would discourage exploratory testing. We are not suggesting that this should be completely unstructured and ad hoc activity, since it should be possible to plan much of the safety verification effort ahead of time if a thorough analysis of the hazards has been performed. However, some effort must be set aside to explore behaviours discovered late in development that are “not non-conformant” and appear to be relevant to safety.
5. The “customer” for the safety verification is the person responsible for writing the final system safety report. He or she determines whether the test evidence generated by safety verification is adequate for the purposes of stating conclusions about the safety of the system. The amount of time and budget provided in support of this effort is decided separately by project management.

Safety Testing Inputs

If safety verification is to be more substantial than re-verification of the safety requirements, then the input into safety testing must be more than just safety requirements.

First of all, safety verification must take into account other requirements that have not been explicitly labelled as “safety requirements” but nevertheless may have a bearing on the possibility of a mishap. The safety requirements of a software-intensive system are often limited to the specification of “safety functions” intended to mitigate hazards by providing some form of protection. For example, this might be a function to activate a fire suppression device when intense heat and smoke is detected. But many other behaviours of the system may have a bearing on the likelihood of creating the physical conditions for ignition of the fire. Although some approaches to software safety are (unfortunately) preoccupied with using protection systems to mitigate hazards, it is also very important to consider what can be done to eliminate the hazard or at least reduce the likelihood of needing to rely on the protection systems to limit harm. Especially with the increasingly integrated nature of software-intensive systems, the safety relevance of certain requirements may not be understood at the time when a subset of the requirements are labelled as “safety requirements”. Their relevance will hopefully be revealed by thorough analysis of hazards. We can assume that these other requirements will be verified by the normal requirements-based testing process. Nevertheless, these other requirements will be highly relevant inputs to hazard-driven safety testing.

Another extremely useful source of input for hazard-driven safety testing are problem reports that were raised during requirements-based testing (or other forms of testing) but rejected on the basis that the observed behaviour has been deemed compliant with the requirements. As observed earlier in this paper, conformance with the requirements does not necessarily imply that the behaviour is safe. Especially if the test engineers have been made aware of the hazards identified for the system, they are likely to raise a problem report as a means of recording a concern that may be relevant to one of the identified hazards. As part of the hazard analysis process, the database of problem reports should be regularly searched for evidence of concerns recorded by test engineers. Problem reports that describe safety concerns may even provide a test scenario that can be adapted for the purpose of safety testing.

Of course, the main input to a hazard-driven approach to safety testing will be the hazard analysis. In approaches to software safety styled on MIL_STD 882 (ref. 1) or like-minded standards and guidelines, hazards provide the framework for the safety analysis effort and documentation of the hazard analysis serve as the main repository for understanding of the system from a safety perspective. Hazards should be defined as high-level safety concerns in terms sufficiently general to encompass a variety of possible hazard causes. For example, “loss of braking function” is an example of a potential hazard for an anti-lock braking system. “Incorrect altitude information displayed” is an example of a hazard that might be identified for a software system used by an air traffic controller. As the main repository for understanding of safety information about the system, the documentation of the hazard analyses should identify both the relevant “safety requirements” as well as other requirements that may also be relevant.

The analysis of each hazard may involve use of common hazard analysis techniques such as Fault Tree Analysis (FTA) and Failure Modes and Effects Analysis (FMEA). FTA is a “top down” technique, beginning with the hazard as the “top event” as the result of a cause-effect relationship. By tracing “backwards” from effects to causes, we discover intermediate causes of the hazard.

These intermediate causes are combined in the FTA using logical relationships, namely “AND” and “OR”. Each branch of the FTA terminates with an internal or external condition. FMEA is a “bottom up” technique, typically used to examine the consequences of failures in system components. It is used in a “forward” fashion, beginning with a potential component hazard cause and then tracing the effects forward to system outputs. With Failure Modes Effects and Criticality Analysis (FMECA), the criticality of the failures and mitigating provisions are also identified.

The results of these specialized analysis techniques may be directly useable as input to the safety verification effort. For example, a test scenario for a particular safety test might be derived by following one branch of a fault tree from a leaf node to the root node – or from several leaf nodes if the fault tree contains AND branches. If the leaf node of a fault tree for a software system refers to an internal condition, it is likely that the hazard analysis will record why this internal condition is impossible or highly unlikely (or else the possibility of this internal condition should be regarded as an unresolved safety problem). However, the test engineer should challenge this conclusion by attempting to force an occurrence of this impossible or highly unlikely internal condition. To this end, the test engineer should use knowledge of the system vulnerabilities and limitations as well as create external conditions that are conducive to an occurrence of the hazard.

For example, the analysis of the “Incorrect altitude information displayed” hazard for an air traffic control system will search for possible causes of a situation where the displayed altitude of a flight is inaccurate. One branch of the FTA will explore scenarios in which altitude information for the flight ceases to appear in the updates generated by the radar data processing system. When altitude information is no longer available, this information should be removed the display or an indication provided to show that the altitude information is stale. Failure of the software to recognize when the altitude information is stale could result in the display of inaccurate altitude information. If sufficiently detailed, the FTA may identify the specific software mechanism responsible for detecting the cessation of altitude information for a flight. Close examination of this mechanism may reveal that correct operation of this mechanism depends on certain implicit assumptions about the manner in which altitude information may be terminated. The safety test engineer should explore possible ways in which these implicit assumptions may not be satisfied. In doing so, he or she may be able to find a particular test scenario that results in an occurrence of the hazard under operationally realistic conditions.

Although the hazards identified by the hazard identification task early in development should provide “homerooms” for most safety concerns identified during development, there are still likely to be a number of miscellaneous safety concerns that surface during development that do not fall within the scope of any identified hazard. In some cases, it may be appropriate to introduce a new hazard if there are a number of related safety concerns that have sufficient mass to warrant a new hazard. But for the remaining concerns, there should be a mechanism for recording and tracking miscellaneous safety concerns that fall outside the scope of an identified hazard. This mechanism could be a database similar to the database typically used to record and track problems related to requirement conformance. This database is another important source of input for safety testing. Moreover, the purpose of this database might even be extended to record and track problems that arise during hazard-driven safety testing.

Safety Testing Focus

Safety testing is intended to supplement the existing requirements-based testing. As noted earlier, demonstrating that a particular requirement, even a safety requirement is “satisfied” does not mean

the safety risk associated with that functionality has been adequately mitigated. Safety testing can provide additional evidence of the successful implementation of the safety requirements based upon the hazard analysis. Ultimately, however, the goal of safety testing is to identify hazard occurrences and not their absence.

It is important to prioritize the safety testing effort based on the results of the safety analysis. Toward that end, the focus of the testing is on high-risk areas that are not adequately covered by the existing testing. In particular, there will likely be “holes” in the test coverage with respect to safety, as identified in the safety analysis. Though the requirements might be demonstrated to be satisfied, further testing is required to determine if the hazard can occur.

A “hole” in the test coverage does not necessarily mean there are no test cases in a given area. Rather, the area needs to be examined in greater depth to uncover potential hazard occurrences. This requires an “inquisitorial” attitude that probes an area from all possible angles, searching out ways in which a hazard can occur. In particular, for high-risk areas, an aspect of the testing will be exploratory. As noted previously, this style of exploratory testing is typically not encouraged in requirements-based testing.

In the case of the “Incorrect altitude information displayed” hazard, there might very well be requirements involving the correct display of altitudes and an indication when the values are stale. These requirements could then be demonstrated with scenarios involving typical radar track updates and the cessation of radar track updates. However, the safety analysis will typically uncover various scenarios in which the altitude information could be lost and the requirements-based testing is unlikely to cover all of them. These might include situations such as the radar updates continuing, but the altitude information is lost due to aircraft transponder failing to transmit the altitude. This is not necessarily a sign of poor requirements or acceptance testing. The requirements may simply express the loss of altitude information at a high level and not specify in detail all the ways in which this occur.

Safety Testing Technique

Exploratory testing of a high-risk safety hole in the test coverage can be performed in a systematic fashion. One approach is to begin with a known occurrence of the hazard, possibly under operationally unrealistic circumstances or circumstances where the behaviour is anticipated and accepted. Once the hazardous behaviour has been demonstrated, the test scenarios are repeated in an iterative fashion, where the unrealistic conditions are gradually modified to become more realistic. The goal of this iterative process is to find an operationally realistic set of conditions that results in an occurrence of the hazard.

The analysis of a particular hazard may reveal different combinations of external conditions that could result in an occurrence of this hazard. This analysis may prompt the addition of changes to the design to provide further mitigation of the hazard. But some combinations of these external conditions may be deemed impossible or extremely unlikely – and for this reason, are not mitigated by design changes. Our approach focuses on these impossible or extremely unlikely conditions. Through exploratory testing, we re-examine the earlier determination that these conditions are impossible or extremely unlikely.

For example, another air traffic control hazard involves the miscorrelation of two radar tracks, where one is mistaken for the other by the system. For situations where there is no altitude or

aircraft transponder code to distinguish the tracks, the radar updates are correlated to a track based upon the track position, speed and bearing. In this case, a system limitation would be the inability to distinguish between two tracks in close proximity both traveling at a similar speed and bearing.

The testing of the hazard could begin by exploiting this limitation with an extreme scenario, i.e., two aircraft traveling in formation wingtip to wingtip, to see if the hazard occurs. If it does, the test is then repeated in an iterative fashion, where the test parameters are systematically modified with each run. For example, the vertical separation could be steadily incremented in one set of test runs, while the lateral separation increased in another. In this fashion, the situations where miscorrelation can occur are explored. The results are then assessed for the safety risk. Though the initial scenario might be dismissed as unrealistic or an acceptable limitation of the system, by exploring the extent of the limitation, an informed safety assessment can be performed.

Summary

It is not enough to merely verify the safety requirements of a safety-related software system. There must also be a hazard-driven test effort that uses the details of the hazard analyses. Although the scope and depth of hazard-driven safety testing can be largely determined from a sufficiently thorough hazard analysis, at least some portion of this effort will be exploratory.

References

1. MIL-STD-882D, Standard Practice for System Safety, Department of Defense, 2000
2. RTCA DO178-B, Software Considerations in Airborne Systems and Equipment Certification, Washington: Radio Technical Commission for Aeronautics, 1992.

Biography

Jeffrey J. Joyce, Ph.D., Department of Electrical and Computer Engineering, University of British Columbia, 2356 Main Mall, Vancouver, BC, Canada V6T 1Z4, telephone - (604) 822-7281, facsimile - (604) 822-5949, e-mail – jeffj@ece.ubc.ca, website – www.ece.ubc.ca/~jeffj.

Dr. Joyce is an Associate Professor at the University of British Columbia (UBC) in the Department of Electrical and Computer Engineering. Before joining UBC, he was a senior systems engineer for Raytheon Systems Canada Ltd, involved in the safety analysis of the Canadian Automated Air Traffic System (CAATS) as well as a related military system (MAATS). His research interests include software requirements specification, software testing and software safety. He received his Ph.D. from the University of Cambridge in 1990, with earlier degrees from the University of Calgary and University of Waterloo.

Ken Wong, Department of Electrical and Computer Engineering, University of British Columbia, 2356 Main Mall, Vancouver, BC, Canada V6T 1Z4, telephone - (604) 279-5783, email - kcwong@ece.ubc.ca

Ken Wong is a system engineer with Raytheon Systems Canada Ltd., and has been involved with various aspects of critical system development, including hazard analysis and safety-related testing. He is also a PhD candidate at UBC, conducting research in the area of software safety.