Mälardalen University Licentiate Dissertation 320

Safety-Critical Software - Test Coverage vs Remaining Faults

Johan Sundell

Mälardalen
University

Mälardalen University Press Licentiate Theses
No. 320

SAFETY CRITICAL SOFTWARE - TEST COVERAGE VS REMAINING FAULTS

Johan Sundell

2022

School of Innovation, Design and Engineering

ISBN 978-91-7485-546-3
ISSN 1651-9256
Printed by E-Print AB, Stockholm, Sweden

Safety-Critical Software - Test Coverage vs Remaining Faults

Licentiate Thesis

Johan Sundell

Neldus Tech AB

johan.sundell@neldustech.com

Supervisors: Kristina Lundqvist, Håkan Forsberg

Company mentor: Jinghong Sundell

Abstract

Safety-critical software systems have traditionally been found in the aerospace-, nuclear- and medical domains. As technology advances and software complexity increases, such systems can be found in more and more applications, e.g. self driving cars. These systems need to meet exceptionally strict standards in terms of dependability. Proving compliance is a challenge for the industry. The regulatory bodies often require a certain amount of testing to be performed but do not require evidence of a given failure rate (which for software is hard to deal with compared to hardware). This Licentiate thesis discusses how to quantify test results and analyses what conclusions can be drawn from a given test effort, in terms of remaining faults in the software.

Sammanfattning

Säkerhetskritiska system är system som kan vålla svåra konsekvenser (förlust av liv, miljöförstöring, mm) om de inte fungerar som de ska. Dylika system, som måste vara extremt tillförlitliga, har funnits i årtionden i flyg-, kärnkrafts- och medicinska tillämpningar. Den snabba tekniska utvecklingen leder till att alltmer komplexa mjukvarusystem, ofta säkerhetskritiska, nu återfinns i alltfler applikationer, t.ex. i självkörande fordon. Sådana system måste uppfylla extremt högt ställda krav på tillförlitlighet. För industrin är det problematiskt att visa att man uppfyller kraven. Myndigheterna kräver vanligtvis att en viss mängd testning ska utföras innan systemen driftsätts, men kräver inte bevis på att en given tillförlitlighet ska visas (vilket anses extra svårt för mjukvarusystem). I denna licentiatavhandling diskuteras hur testresultat kan användas för att statistiskt kvantifiera mängden kvarvarande fel i ett system. Syftet är att bättre kunna förutse tillförlitligheten innan driftsättning.

Dedicated to Jinghong
David, Isak, Selma and Celia

List of papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.

I Model of Fault Distribution in Complex Safety Critical Systems

II Prediction of Undetected Faults in Safety-Critical Software

III Safety-Critical Software - Quantification of Test Results

Reprints were made with permission from the publishers.

Contents

1. Introduction

Software based applications have been changing our daily lives. We wake up every morning to our a digital clocks; we stream the morning news on our smart TVs; our cappuccino is made by automatic coffee machines; our movements and steps are monitored by our smart watches; on our drive to work we monitor the traffic situation on our smart phones; on the way we listen to music provided by streaming services; we use our smart phones to call and communicate with friends, colleagues, customers, and to learn what has happened in the world. Software is so integrated in today's world that we hardly reflect on it.

This development has made our lives more convenient and has relieved humans from doing many routine tasks as well as dangerous activities like mining or tasks requiring exposure to radiation or chemicals. The capabilities of software is developing rapidly with the emergence of areas like AI (Artificial Intelligence), ML (Machine Learning) and IoT (Internet of Things). However, this means that more errors and bugs will come along. More and more complex software poses a challenge for software testing. For simpler software, e.g., a music streaming service, sufficient testing can be to make sure that the product can perform a certain task, like select a chosen tune. If there are a few cases where this does not work, it can obviously be annoying but never dangerous. The scope of the testing of such systems have to be extensive enough to make sure that the system lives up to the user/customer expectations of the product, i.e., it cannot have too many annoying bugs.

Bear in mind that testing software is different from testing activities in general, in that it is binary by nature. There are so called "cliff effects" which means that one input can be safe and an almost identical input can trigger a failure. In many cases there is no warning or indication that the system is close to fail. This peculiarity of software underlines the importance of good test coverage when high reliability is required.

For safety-critical systems, especially those that undergo a certification process, the scope of the testing is a different matter. Safety-critical systems are often defined as systems that can cause serious consequences if they fail, e.g., loss of lives, environmental damage, financial loss. Such systems need to be extremely reliable and fulfill stringent requirements, e.g., the aviation standard DO-178C [28] requires a failure rate of less than 10^{-9} faults per flight hour, for systems that can cause a catastrophic failure. This is equivalent to a failure rate of one failure per approximately 114,000 years.

The problem is that there is no way of proving that such requirements are fulfilled. Even if testing is accelerated it is virtually impossible to show compliance [10, 22]. However, since this is known and accepted by the certifying bodies, an alternative method has been adopted. Instead of providing proof that the such requirements have been fulfilled, it is sufficient to provide proof that best practices have been followed throughout the development process. What constitutes best practices is domain dependent and described by accepted standards, e.g., ISO 26262 [18] for the automotive domain, DO-178C [28] for the aerospace domain and, IEC 61513 [17] for the nuclear domain.

A closer look at the history of the traditional safety-critical domains (aerospace, nuclear, medical) shows that this approach has worked out fairly well, i.e., very few accidents can be attributed to faulty software over the years. There are some well known exceptions where software has been a main cause of substantial damage, e.g., Therac-25 medical radiation equipment [20], Ariane 5 rocket crash [23], Knight Capital $440 million bug [1] . Additionally, there are a number of classified occurrences in the military domain.

However, the above mentioned "mature" domains have had the chance to gradually build experience of developing safety-critical software. Procedures and practices have, for decades, had time to evolve with the capability of technology and the number of systems in use have been limited. Meaning that, there are relatively few nuclear power plants, medical radiation equipment, and a typical fleet of aircraft numbers in the hundreds. But, the current technology development means that this situation is now changing rapidly. The exposure of safety-critical systems will increase dramatically when such systems are implemented in new domains, e.g., self-driving cars. Bertolino, et al., show in [7] that a critical system in a typical fleet of self-driving cars statistically have a mere 5% probability of surviving its lifetime without encountering a catastrophic failure, somewhere in the fleet. They compare this with a critical system in an aircraft fleet which has a 99.64% probability of surviving. These numbers assume that the actual failure rate fulfills a requirement of 10^{-9} failures per execution. If this is not the case, i.e., the actual failure rate is higher, the situation can be orders of magnitude worse and a significant increase of software caused catastrophic failures can be expected. Increased exposure in combination with the difficulties of predicting the failure rate prior to deployment, is problematic.

The point here is that the expected massive exposure of safety-critical systems in the near future will undrape any weaknesses in the current way of certifying such systems. The current gap between the failure rate that can be claimed by following a certification process and what can be statistically proven, does not inspire confidence. McDermid calls this the "prediction gap" [24]. In order to move on and enable the technological development in this area it is important to develop and improve methodologies that allow confidence building and assessment of software failure rate prior to release, i.e., reduce the "prediction gap".

This Licentiate thesis is addressing this objective by improving the interpretation of test results. The main idea is to view test results not just as being a fault removal activity to fulfill a process but also as a statistical sample of the software, from which conclusions of remaining faults can be drawn.

Throughout this thesis, the following definitions are used:

- Model: The model defined in Paper I.
- Fault: It is a condition that causes the software to fail to perform its required function. (IEEE)
- Failure: It is the inability of a system or component to perform required function according to its specification. (IEEE)
- Failure density: The risk of a random input triggering a failure. From Paper II.

1.1 Problem Statement

The capabilities of software is developing rapidly with the emergence of areas like AI (Artificial Intelligence), ML (Machine Learning) and IoT (Internet of Things). This development can relieve humans from doing many routine tasks but also dangerous activities like mining or tasks requiring exposure to radiation or chemicals. There are many areas where there is a great potential for improving safety by letting software do the job instead of humans. CFIT constitutes around 6% of the aviation accidents [16]. CFIT is an acronym for Controlled Flight Into Terrain, which is an in-flight collision with terrain, water, or obstacle without indication of loss of control, i.e., the pilot flies the aircraft into the terrain due to lack of situational awareness. Another example from the automotive domain is driver error, which is the leading cause of road accidents [2]. A higher degree of automation is seen as a possible solution to such problems.

In the past such safety-critical systems have been found in a few domains such as aerospace, nuclear and medical. These domains have over the years acquired a certain level of maturity when it comes to development of safety-critical systems. Standards, e.g., DO-178C [28] or IEC 61513 [17], have evolved to give guidance on best practice.

However, there has been the lingering problem of actually proving the target failure rates prior to deploying the systems. McDermid [24] talks about a "prediction gap", between what is claimed and what is proven.

The size and complexity of today's systems rules out exhaustive testing and leaves a doubt whenever a new system is deployed.

For safety-critical software there is a lack of methods and strategies to deal with the issue of predicting the failure rate or even, with confidence, determine the status in terms of safety. The drive towards autonomous vehicles, advanced IoT, artificial intelligence etc, means that this problem is further aggravated.

Introduction of formal methods (mathematical techniques) have shown promise but have until now struggled to gain a wider industry acceptance. Random testing is another way to improve the knowledge of software based systems and to achieve a significantly improved test coverage. However, in order to maximize the benefit of increased testing it is important to quantify what the increase means, in terms of improved quality/safety.

This Licentiate work addresses the problem of a lack of methods for interpretation and quantification of test results.

1.2 Research Objectives

The scope of this work can be described as having the purpose of reducing the so called "prediction gap"[24]. The work is divided in two parts, the Licentiate and the PhD.

The first part, which comprise the Licentiate thesis, is theoretical and is focused on quantifying the remaining faults in the software after tests. This requires an improved modeling of fault detection vs test coverage.

The second part, is more practical and has the purpose of increasing the test coverage, by applying massive structured random testing on commercial industry software.

Scope of the work (Licentiate and Phd):

1. Quantification of test results
   This part of the work analyzes the effect of test effort. The analysis is focused on which conclusions that can be drawn from a given test coverage, and test result.
   In order to achieve industry acceptance, predictions need to be made using parameters that are known from the test process.
2. Distributed Autonomous Random Testing
   This is a practical implementation of Autonomous random testing. Focus is on defining a methodology for an autonomous testing that does not require user supervision or interaction. This allows for utilization of redundant computational resources connected to the network. By using the idle computational power it is possible to achieve a significant increase in the pre-deployment test coverage.

1.3 Thesis Objective (Licentiate)

The overall objective of this Licentiate thesis is to address the problem of quantification of undetected faults in safety-critical software.

The objective is met in the following ways:

1. Define a model that describes random testing (of safety-critical software) in terms of fault triggering.
2. Define a method for prediction of the remaining undetected faults.
   In addition, the input to the equation must be parameters that are available in an industry setting.
3. Investigate limitations of the prediction.
   Theoretical analysis
   Simulated data is analysed. The initial results are presented in paper I and a generalized form of the prediction equation is presented in paper II.
   Analysis of industry test data
   In this work a tool is implemented to run long test suites. The results from the test suites are used to calculate the detected and undetected faults of the system. These results are checked for consistency.

1.3.1 Research questions

The research questions have been iteratively refined, in accordance with the steps described in section 2 Research process.

The following research questions and their rationale have been defined:
M - Model related
Ind - Industry related

1. M-RQ1: How can the fault finding dynamics of random testing of safety-critical systems be modeled?
   A model is required for gaining understanding of the subject.
   M-RQ1a: Which factors affect the amount of remaining faults?
   Understanding what affects the final result is vital for improving the testing process.
   M-RQ1b: Can industry test result data be used for prediction of the amount of remaining faults?
   Industry acceptance can only be achieved if such data can be used.
2. Ind-RQ2: Can the required random (uniform) sampling strategy be implemented in an industry setting?
   The prediction equation requires sampling of the full input space, which needs to be fully identified and analyzed.
3. Ind-RQ3: Can the amount of remaining faults be predicted when the test coverage is limited?
   It is important to investigate if it is possible to make predictions in those cases were sampling of the full input space is unfeasible.

4. Ind-RQ4: How can the concept of (failure density based) entropy (Shannon information) be implemented in an industry software testing process?
   Shannon information (entropy) is a measure of uncertainty that fits well with determining the status of safety-critical software. Guidance for industry implementation is needed.

1.3.2 Assumptions

The context of this work is testing complex safety-critical software. It deals with quantification of the effect of a substantial amount of testing. The characteristics of safety-critical software differ from software in general in that it is more reliable, deterministic, and has less faults. These properties make it possible to approach the problem of quantification from a new angle. The rationale of the assumptions is discussed in more detail in Paper II.

For this work below assumptions are made.

The main assumption is:

∗ There is a relationship between the length of the test suite and remaining undetected faults.

Below assumptions come from the nature of safety-critical software. They may not be valid for software in general, but are consequences of the way this software is developed and designed. The development standards, e.g., DO-178C [28], require that all code must reachable and executed during testing, i.e., unreachable code, dead code must be removed. To increase the safety a number of practices, sometimes called defensive programming, are used to prevent code from executing unintended or unpredictable operations, e.g., constraining the use of certain structures, avoiding unsafe numerical operations, and not allowing dynamic memory allocation.

∗ The input- and state spaces of the system are reachable.

∗ All parameters are defined at runtime.

∗ The system has a deterministic behavior.

Other assumptions that are made in this work, should be seen as limitations describing the focus of the performed simulations or have been derived during the research process. The use of uniform sampling is a deliberate step away from assuming a user profile, the purpose is to achieve a more robust predictability. This eliminates questions about the accuracy and correctness of the user profile, i.e., a small deviation between actual and assumed user profile can lead to very different predictions. The minimum number of faults should be seen as stating that this work is focused on more complex systems. According to a study, funded by the UK MoD, the software of transport aircraft C130J was found to have 1.4 safety-critical faults per 1,000 LOC (lines of code) [13]. McDermid states [24] that 1 fault per 1,000 LOC can be seen as world class. This means that an assumption of at least 50 faults translates to a code size of at least 35-50 kLOC, which excludes smaller systems.

- Uniform sampling of the input space is used.
- More than 1,000 test cases are generated.
- More than 50 faults are present in the system prior to testing.

2. Research Process

This section presents an overview of the methods that we used to conduct the research. We first describe the general process that we follow in our research, after which we explain the selected methods used in this thesis. Our research process is shown in Figure 2.1.

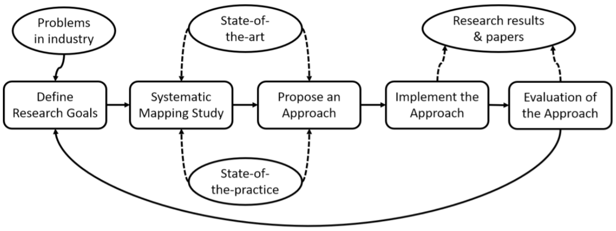


Figure 2.1. Overview of research sequence, -activities and -results.

This work started with an interest in investigating the relationship between test coverage and remaining undetected faults in safety-critical software. The lack of methods for proving that safety requirements are fulfilled or even estimating the safety of the software, became evident at a point when our industry team (in the aerospace domain) had been able to dramatically improve the test coverage. This was achieved by adding a combination of random testing and by preforming a large number of static test cases. The results were obvious to everyone involved, bugs could now be found earlier in the development, previously undetected bugs surfaced, and the development process became more predictable (on time). However, the overall improvement could not be quantified in terms of how much safer had the software become from the performed improvements.

The relationship between test coverage and software safety has been hard to describe, both for the industry and academia. This area of research has been seen as a futile part of computer science. Quotes like "Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence" (Edsger Dijkstra) [12], have long been the common view of this problem.

Based on the assumption that there indeed is a describable relationship, research goals were formulated. In order to address these goals and determine state-of-the-art and state-of-practice, the critical analysis of literature method [15] was used.

The selected systematic approach was determined to be an analysis of the problem, by the method of Mathematical modeling [15]. The created model was explored using simulation. The results from the implemented approach was reported in reports and research papers, see Papers I - III.

The results determined if new refined research goals should be defined or if a new literature search should be performed and a revised proposed solution should be defined. During this research process a number of research methods were applied. Through the initial stages the method Proof of concepts [15] was used to show the correctness and applicability of our proposed approach.

Finally we applied the method of Proof by demonstration [15], by developing a tool that provided the necessary input for analysis of the proposed prediction equations.

3. Thesis Contributions

The following are the main research contributions included in this thesis, collected from the set of included papers.

The full PhD-thesis is investigating ways to reduce the so called "prediction gap", between proven and claimed (expected) in-service failure rate for safety-critical software. The research strategy is to significantly increase the number of performed tests and to improve the interpretation of the test results. The practical part of this work is implementation and optimization of autonomous testing, a continuously running test program that defines, runs and evaluates test cases. The result of this is a radically increased test coverage.

The theoretical part is investigating how to quantify this increase by describing the relationship between software quality and length of the test suite. The Licentiate part of the PhD-thesis is focused on the theoretical part.

The following are the main research contributions included in this thesis, collected from the set of included papers.

- **Model of the effect of random testing (Paper I)**
  In this paper a theoretical 3D model is introduced. By studying the effect of parameters such as length of test suites, and initial distribution of faults, the model allows for a visual representation of the probability of fault detection. This model answers the research question *M-RQ1: How can the fault finding dynamics of random testing of safety-critical systems be modeled?* and the theoretical work in all papers use this basic model.
- **Definition of prediction equation for long test suites (Paper II)**
  This work looks at the special characteristics of safety-critical software that allows for quantification of the undetected faults remaining in the system. The paper also describes why the use of uniform sampling is preferred. The test results, in terms of number of tests, number of faults triggered, and number of identified unique faults, are used as a statistical sample of the software. The sample is used to predict the sum of the undetected faults. The research questions that are answered and reported in Paper II are; *M-RQ1a: Which factors affect the amount of remaining faults?* and *M-RQ1b: Can the data from an industry test result be used for prediction of the amount of remaining faults?*
- **Definition and validation of the use of entropy as a test metric (Paper III)**
  This work presents a generalization of the previous equation in Paper II. Here the theoretical model is validated against industry data. It addresses

the cases in the industry where an extensive sampling of the input space is infeasible or too costly. The paper shows how to predict the failure density with a significantly lower coverage of the input space. Furthermore, it discusses that for more complex software, a uniform sampling strategy may not be sufficient in an industry setting. For example, if the system processes a number of subsequent inputs like control loops, it may be necessary to apply a so called Grey box approach, i.e. to analyze and adapt the sampling strategy to the software structure. Two research questions are answered in this part of Paper III; *Ind-RQ2: Can the required random (uniform) sampling strategy be implemented in an industry setting?* and *Ind-RQ3: Can the amount of remaining faults be predicted when the test coverage is limited?*
Moreover, Paper III shows the use of entropy or Shannon information as a test metric, which has been suggested by for example Clark [11]. Entropy can be seen as a measure of "ignorance" of the system. This "ignorance" can be reduced by the test process. The paper shows how this concept can be implemented in an industry setting. The paper also discusses the dynamics and limitations of fault detection and shows how important fault size distribution is to the end result, i.e., remaining faults. This part of the paper answers the research question; *Ind-RQ4: How can the concept of (failure density based) entropy (Shannon information) be implemented in an industry software testing process?*

3.1 Model of safety-critical software testing

A model needs to describe the key aspects of what is being researched. In order to be a tool for making predictions, a good model needs to be both accurate and simple. However, even a good model will almost always have limitations.

The defined model, though only internally published [29], has been of great importance to this work. Its purpose is to describe the effect of a randomly generated test suite of a given length (number of test cases) on remaining undetected faults. The model has been vital for exploring and gaining an understanding of the dynamics of safety-critical software testing. It is important to understand what is modeled, as it is not generic software testing. In this section the following aspects will be discussed further, the uniform sampling strategy and n-dimensionality, the concept of volumes representing risk of being triggered (fault sizes), and factors affecting the remaining undetected faults.

Safety-critical software testing differs in some important aspects from software testing in general. The input space of a safety-critical system is restricted by validity and feasibility checks, i.e., out-of-range values are not accepted as valid and therefore not considered by the system. Typically, such out-range-values are handled by separate code, that also need to be tested. How to implement and optimize the testing to cover also the error handling code will be discussed further in later papers. The limitation of the input space allows for sampling of, and drawing statistical conclusions related to this finite input space, e.g., 0.5% of all input can be expected to trigger failures. Another aspect that is different is that unreachable code is not allowed. As a consequence the full input space must cover all the code.

3.1.1 Random (uniform) sampling

The model assumes that random (uniform) sampling is used, i.e., all parameter values are equally probable. Moreover, as is required for safety-critical software, all parameters are initialized and defined. Uniform sampling is a different approach from the previously suggested, definition and focus on an expected user profile, see section 5 Related work. This means that a large number of unrealistic test cases, with unlikely parameter combinations are performed. However, there is a point here, this uniform sampling and variation of all parameters makes it possible to explore the full input space, regardless of the number of parameters.

Example:

Let us assume a simulation scenario where a truck is driven on a highway at high speed. An input vector is randomly generated and all parameters are defined. The ignition switch has four positions (possible values); OFF, ACCESSORY, RUN, and START. It might seem strange that all four values are equally likely to be selected, but that is what uniform sampling of the full input space represents. It should be noted that the oracle (the evaluation of the output/result) needs to take into consideration, the expected behavior. In this case, loss of power may be a normal and expected behavior for the OFF position, while considered a failure for the other possible positions.

3.1.2 Volumes and fault sizes

An important concept that the model needs to capture is that different faults occur at different rates, i.e., they are more or less likely to be triggered by (in this case) a random input.

Littlewood and Strigini state in [21]; *A plausible conceptual model of the software failure process is as follows. A program starts life with a finite number of faults, and these are encountered in a purely unpredictable fashion. Different faults contribute differently to the overall unreliability of the program: some are "larger" than others, i.e., they would show themselves (if not removed) at a larger rate. Thus different faults have different rates of occurrence.*

The concept of fault size is implemented in the model. Thus, a large fault means that a relatively large partition of the input space represents a specific fault. This is equivalent to saying that it occurs at a relatively large rate during random sampling of the input space.

Figure 3.1 shows the faults of the system modeled as subvolumes. The (size of the) subvolume is proportional to the risk of being triggered by a random input. This should be viewed as a number of parameters, typically less than six [19], forming a volume within the full input space. Within this subvolume, where all parameters are in a critical range, a fault is triggered. The size or risk of being triggered is the only aspect that the model needs to capture. The positions and shapes of the subvolumes are therefore of no importance in the graphical representation. The model randomly generates independent subvolumes. They can, just as in real software, overlap.

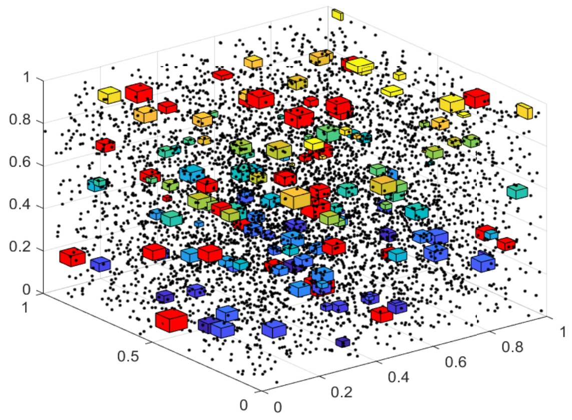


Figure 3.1. 3D model of safety-critical software testing. The N-dimensional input space is projected onto a 3D volume. The black dots represent performed test cases. The red boxes are triggered (detected) faults. All other boxes depict the (so far) undetected faults.

The test cases are seen as black dots in Figure 2.1 and represent a randomly generated fully defined input vector. Moreover, the resulting triggered or not triggered fault, fits well with the typical safety-critical evaluation of anomalies. The standards require a binary classification of all findings as being OK or NOT OK. There is no grey zone.

Example:

Let us look at an Automatic Braking System for a car, that engages when there is an imminent collision. Let us assume that the system should be triggered at the situation shown in Figure 3.2. Let us now assume that the system for some reason does not activate, due to a fault in the software. In this case the car is moving at 50 km/h at a distance of 20 m to traffic ahead, which is moving at 15 km/h. If the same fault is also triggered under slightly different circumstances, e.g., 49.2 km/h / 21m, and 16 km/h. We have a case where the parameters span a volume, such as the ones in the model (Figure 2.1).

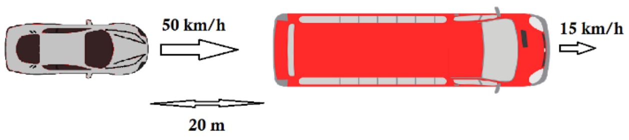


Figure 3.2. Three parameters triggering a fault in an Automatic Braking System of a car

The model allows for changing some initial conditions, i.e., initial sum of faults, number of faults, and which statistical distribution to use for generation. Which distribution to use is a complex matter. Some theoretical papers like Arcuri's [3] use Uniform distribution. This is of course a simplification of the problem that means that all faults are equally probable. Many statisticians argue that the default choice is a Gaussian distribution, which implies that that faults are independent. However, there are some questions related to this choice. Examples, what would be the mean fault size and another which factors affect the mean, and how to handle the negative tail part of the distribution. Other researchers, like Bishop [9], argue for a lognormal distribution. However, they refer to non safety-critical software, e.g., IBM products, that are developed very differently.

The software development process affects the fault distribution of the system. A more stringent safety-critical process aims at reducing remaining faults. It is not determined if such measures also reshapes the distribution and to what degree, e.g., if you remove (more likely larger) values from a Gaussian distribution it will at some point no longer fit as Gaussian. The more stringent the development process, the more the introduction of faults is reduced. The end result can be assumed to be a flatter distribution. The reduced introduction of faults, is not only the effect of early testing on unit- , and module level, but also stems from activities such as reviews and careful requirement management.

3.1.3 Important factors

The process of defining the model lead to some interesting insights. The initial prototype was a 2D model. However, in order to better fit the commonly used space associations (input-, state-, and output space) a 3D model was defined. It turns out that the dimensionality does not matter in this case. The simulation results are the same for 1D, 2D, 3D and 4D (moving volumes), as the fault sizes in all cases are proportional to the full length, area or volume.

The conclusion here is that as long as all parameters are defined and uniformly varied during testing, the number of parameters (complexity) of the system does not affect the result.

With the model in place the work with finding a prediction equation could commence. During the simulations all internal model parameters are available, e.g., all individual fault sizes. However, as such parameters are not available in the industry they could not be used for prediction.

The conclusions that soon became obvious were that a few factors are the most important for the final result (the remaining undetected faults).

1. The granularity of the faults.
   Many smaller faults are harder to detect than fewer larger.
2. The initial fault volume.
   The more faults that are in the system from the beginning the more faults will remain undetected for a given test suite.
3. The length of the test suite (number of test points).
   The longer the test suite the fewer the faults that remain undetected.
4. The distribution of volumes/fault sizes.
   The remaining faults depend on how the fault sizes are initially distributed. If the initial distribution have many larger faults or if they are more even in size affects the outcome.

3.1.4 Prediction equation

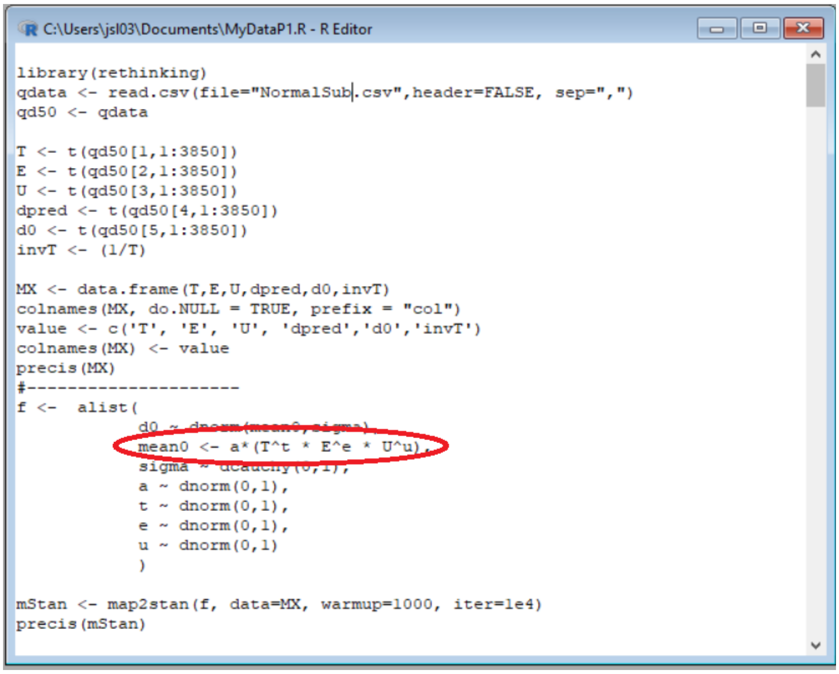
The prediction equation is defined in Paper II:

$$\begin{array}{l}
\text{Number of tests} = \tau \\
\text{Number of failures detected} = \varepsilon \\
\text{Number of identified unique faults} = \upsilon \\
\text{Failure density} = \rho \\
\text{(sum of remaining undetected faults)} \\
\lambda = \upsilon^2/(\varepsilon \times \tau) \\
\rho = \lambda \times e^{-\lambda}
\end{array} \tag{3.1}$$

The equation predicts the sum of the remaining faults, after a given test effort. The result is in this context called failure density, i.e., the risk of a random input vector triggering a fault. The equation considers longer test suites (test cases > 1k), which means that the exponential term is close to 1. However, despite not contributing much to the prediction, the exponential term improves the correlation slightly. The equation predicts well (correlation > 0.9) for a number of fault size distributions, e.g., Normal, Lognormal, Uniform., Geometric.

Correlation and accuracy are reduced for L-shaped distributions, were one (or a few faults) are of a different order of magnitude than the others. In an industry setting this would be manifested by one or a few faults being triggered much more often. The recommendation is to remove the occurrences of such a fault(s) from the calculation.

The set of faults and their fault sizes (risks of being triggered) are distributed in some form. As discussed above in section 3.1.2, the distribution can be assumed to be affected by the software development process and the scope of the testing, i.e., more rigorous process with more testing statistically finds more of the larger faults. This means that the original distribution is flattened. If the underlying distribution was (better) known, the prediction equation could be improved using the Bayesian tools, described in[25].



```
library(rethinking)
qdata <- read.csv(file="NormalSub.csv",header=FALSE, sep=",")
qd50 <- qdata

T <- t(qd50[1,1:3850])
E <- t(qd50[2,1:3850])
U <- t(qd50[3,1:3850])
dpred <- t(qd50[4,1:3850])
d0 <- t(qd50[5,1:3850])
invT <- (1/T)

MX <- data.frame(T,E,U,dpred,d0,invT)
colnames(MX, do.NULL = TRUE, prefix = "col")
value <- c('T', 'E', 'U', 'dpred','d0','invT')
colnames(MX) <- value
precis(MX)
#----------------------
f <-  alist(
            d0 ~ dnorm(mean0,sigma),
            mean0 <- a*(T^t * E^e * U^u),
            sigma ~ dcauchy(0,1),
            a ~ dnorm(0,1),
            t ~ dnorm(0,1),
            e ~ dnorm(0,1),
            u ~ dnorm(0,1)
            )

mStan <- map2stan(f, data=MX, warmup=1000, iter=1e4)
precis(mStan)
```

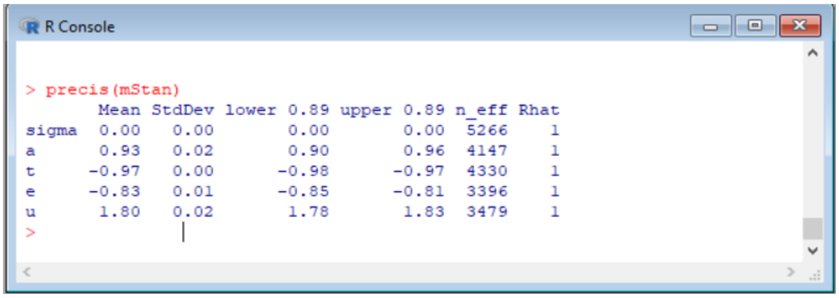
Figure 3.3. R code. The code starts with a command to load the *rethinking* library. This is followed by loading the data from a csv-file and assigning the parameters. The most important are T (τ length of test suite)) , E (ε number of triggered faults), U (υ unique faults), and d0 is the actual (not predicted) result of the simulation, i.e., sum of remaining faults. The program is then asked to fit the data (T, E, U, and d0) with the unknown factor a, and the coefficients t, e, and u.

Example:

Optimization of the equation parameters, through analysis of a dataset containing the results from 3850 simulations of testing of systems with normally distributed fault sizes. Each simulation was started with an initial fault density of up to 0.02 (2%) and a test suite length between 1,000 - 100,000. The results comprise the following data; τ length of test suite , ε number of triggered faults, υ unique faults, and ρ the final failure density. The optimization is a matter of finding an optimum fit for the relationship between τ , ε, and υ on one hand and ρ on the other hand.

The dataset is analyzed using a Markov Chain Monte Carlo (MCMC) tool, *map2stan* from the *rethinking* package. The code is shown in Figure 3.3.

Step one is to load the data from a csv file. Then the *map2stan(...,...)* call is asked to fit the factor *a* and coefficients *t*, *e*, and *u* to the resulting final failure density (d0). The priors are vague, which allows for fitting both negative and positive values.



```
> precis(mStan)
       Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
sigma  0.00   0.00       0.00       0.00  5266    1
a      0.93   0.02       0.90       0.96  4147    1
t     -0.97   0.00      -0.98      -0.97  4330    1
e     -0.83   0.01      -0.85      -0.81  3396    1
u      1.80   0.02       1.78       1.83  3479    1
>
```

Figure 3.4. R console. The resulting fit of *a*, *t*, *e*, and *u* for the dataset given in above Figure 3.3. The values can be read from the Mean column. The other columns list Standard deviation, lower- and upper bounds, n_eff - an MCMC parameter for the number of effective jumps, and Rhat which should converge to 1 for a successful calculation.

After a few hours of computation, the results can be viewed with the command *precis(mStan)*. Note that the results may vary somewhat, as this is a random process. Figure 3.4 shows that the best fit is close, but not identical, to equation 3.2. If the resulting values were rounded you have the original prediction equation, as *a* represents the exponential term (expected to be close to 1) and *t*, *e*, and *u* would be -1, -1, and 2, respectively. However, for this dataset an optimized prediction equation (with minor variations) would be:

$$\begin{aligned}&\text{Number of tests} = \tau\\ &\text{Number of failures detected} = \varepsilon\\ &\text{Number of identified unique faults} = \upsilon\\ &\text{Failure density} = \rho\\ &\text{(sum of remaining undetected faults)}\\ &\rho = 0.93 \times (\upsilon^{1.80} \times \varepsilon^{-0.83} \times \tau^{-0.97})\end{aligned} \qquad (3.2)$$

In this case, the original (not optimized) prediction gives a correlation coefficient of 0.9884, and a mean absolute error of 5.18 x 10^{-4}. The corresponding values for the optimized version are 0.9996 and 5.97 x 10^{-4}, i.e., a minor improvement of the correlation but with a slightly higher mean error.

However, in this case, such minor improvements may be of little interest, as the purpose of the prediction equation is to give an order of magnitude of the remaining faults, by summarizing a large number of test results.

In Figure 3.5, a comparison between the optimized- and original predictions are shown. The results are comparable with few differences.

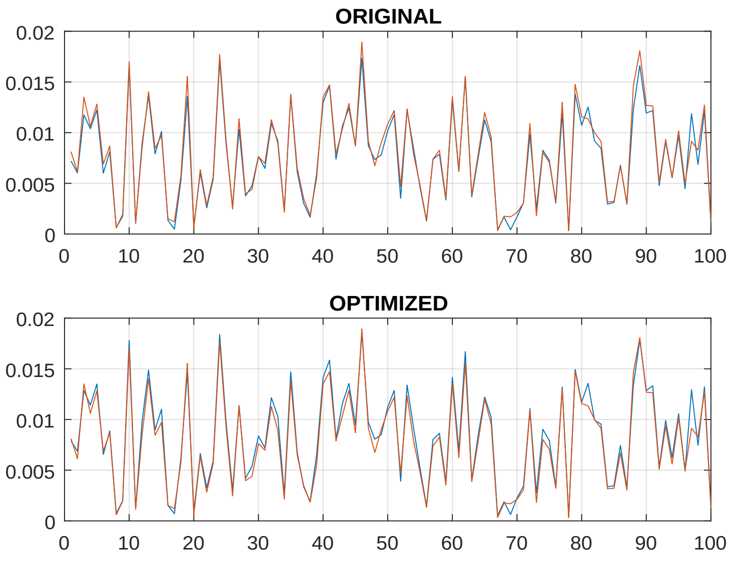
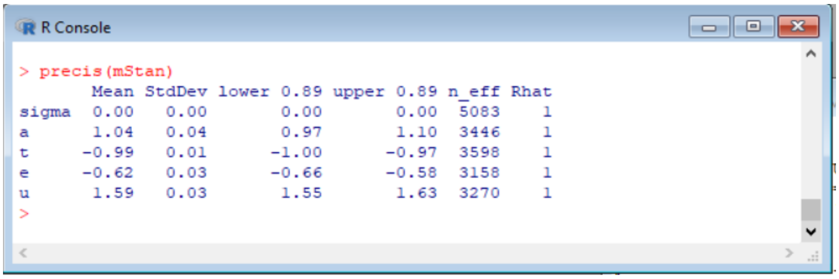


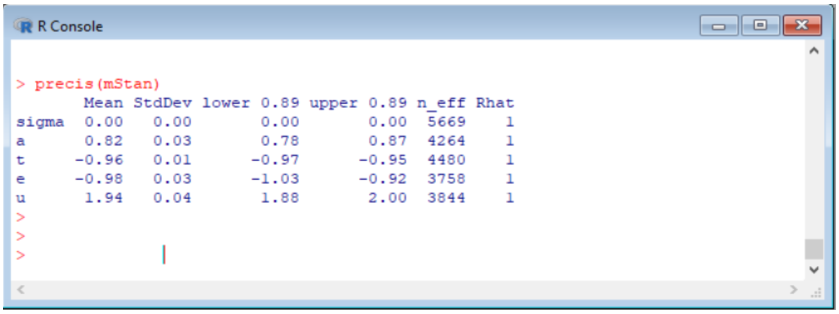
Figure 3.5. Comparison between the original- and the optimized predictions. In the upper plot the original predictions (blue) are plotted togehter with the actual results (red). In the lower plot the optimized predictions (blue) are plotted against the same results (red). In both plots, the y-axis represents failure density and the x-axis simulation sample number. The full vectors are of length 3850, in the plots only values 201-300 are shown.

The same optimization can be done using other fault distributions. Figure 3.6 shows the results of an optimization of a dataset from 1053 simulations with Lognorrmal fault distributions, and Figure 3.7 shows 1040 simulations with Uniform distribution. In both these cases there are minor improvements of the correlations. The mean absolute error for the Lognormal case is improved from 6.18 x 10^{-4} to 4.82 x 10^{-4}. For the Uniform case it is worsened from 5.05 x 10^{-4} to 5.42 x 10^{-4}. However, these relationships are multi-dimensional and dynamic and for more L-shaped distributions the improvements are expected to be greater. The results suggest that for the expected (relatively) flatter distributions, such as Uniform, minor optimizations can be achieved. How to improve the prediction equation is further discussed in section 6.1.



```
> precis(mStan)
       Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
sigma  0.00   0.00       0.00       0.00  5083    1
a      1.04   0.04       0.97       1.10  3446    1
t     -0.99   0.01      -1.00      -0.97  3598    1
e     -0.62   0.03      -0.66      -0.58  3158    1
u      1.59   0.03       1.55       1.63  3270    1
>
```

Figure 3.6. R console - Lognomal. The resulting fit of a, t, e, and u for a dataset with 1053 simulations.



```
> precis(mStan)
       Mean StdDev lower 0.89 upper 0.89 n_eff Rhat
sigma  0.00   0.00       0.00       0.00  5669    1
a      0.82   0.03       0.78       0.87  4264    1
t     -0.96   0.01      -0.97      -0.95  4480    1
e     -0.98   0.03      -1.03      -0.92  3758    1
u      1.94   0.04       1.88       2.00  3844    1
>
>
>
```

Figure 3.7. R console - Uniform. The resulting fit of a, t, e, and u for a dataset with 1040 simulations.

3.2 Conventional approach

The development standards, e.g., [18], [28], [17], define a minimum acceptable test coverage, which results in a given number of tests. All faults identified during these tests need to be addressed. At this stage, the system is considered to be safe enough, as all identified faults have been fixed. However, if one assumes that the identified faults actually comprise all existing faults in the system, the probability of the performed number of tests actually triggering them is close to zero.

Figure 3.8 shows an example of the result from a 5000 tests long test suite. The detected faults are shown by the red bars and the still undetected in blue. Initially, there were 100 faults comprising 1% of the input space, i.e., 100 normally distributed faults comprise 1% of the input space volume. The 5000 tests have detected 48% of the initial fault space (part of the input space).

If we assume that the 37 detected faults were the only faults in the system and we reran a random test suite of the same length, the chance of triggering the 37 specific faults is close to 0.00%. In this case the probability of finding even the 10 largest faults is less than 1.4%. The point is that it is very improbable that all faults have been identified by the performed (5000) tests.

There is a notion that safety-critical software is thoroughly tested, that the use of the Modified Condition/Decision Coverage (MC/DC) criteria is compa-

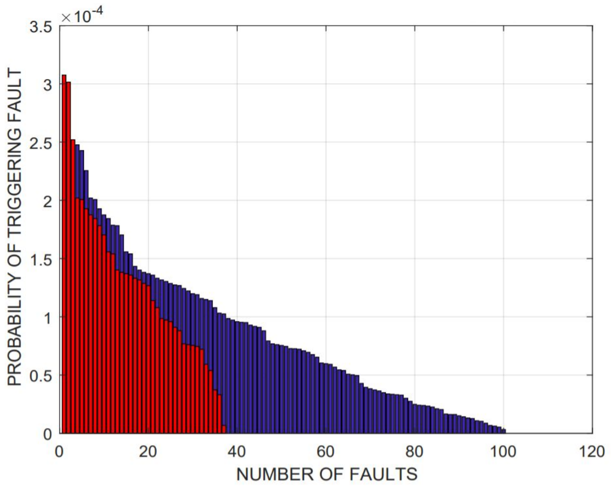


Figure 3.8. In this example, 37 of the 100 faults were triggered during the test suite. Each individual fault has a certain probability of being triggered by a random input, this is sometimes called the “size” of the fault. The red bars represent the faults that have been triggered during the test suite, and the blue bars the remaining undetected. The probability can be read from the y-axis. In this case, the test suite length is 5000.

rable to exhaustive testing. MC/DC-criteria means that each condition in the code needs to be shown to independently affect the decision. However, this is not in any way a guarantee that a significant part of the full input space is covered. For more complex systems, with tens or hundreds of parameters, there is a very large number of possible combinations of system states. This means that the surface of possible outcomes is barely scratched, even though MC/DC-criteria has been fulfilled. For example, if you test a cockpit switch and fulfill the MC/DC-criteria during ground operations, you could miss unsafe system responses elsewhere in the operational envelope. There is no guarantee of a safe system response during take-off, landing, cruise, at high altitude, low fuel, high speed, and so on.

3.3 Alternative approach

This section discusses the reasons for the chosen approach.

Parnas states in [27] *It is difficult to make accurate predictions of software reliability and availability: Mathematical models show that it is practical to predict the reliability of software, provided that one has good statistical models of the actual operating conditions. Unfortunately, one usually gains that information only after the system is installed. Even when a new system replaces an existing one, differences in features may cause changes in the input distribution.*

The uncertainties attached to determining statistical data or a reliable user profile, with precision, makes such an approach unviable. Even a minute difference in the input may effect the output dramatically. Therefore, a different, alternative approach was chosen. The aim was to look at the entire input space and thus be able to reach more robust and objective conclusions.

The input space of safety-critical systems are typically more clearly defined than that of other systems. The reason is that the input is strictly formatted and checked to be within the allowed range before it is used.

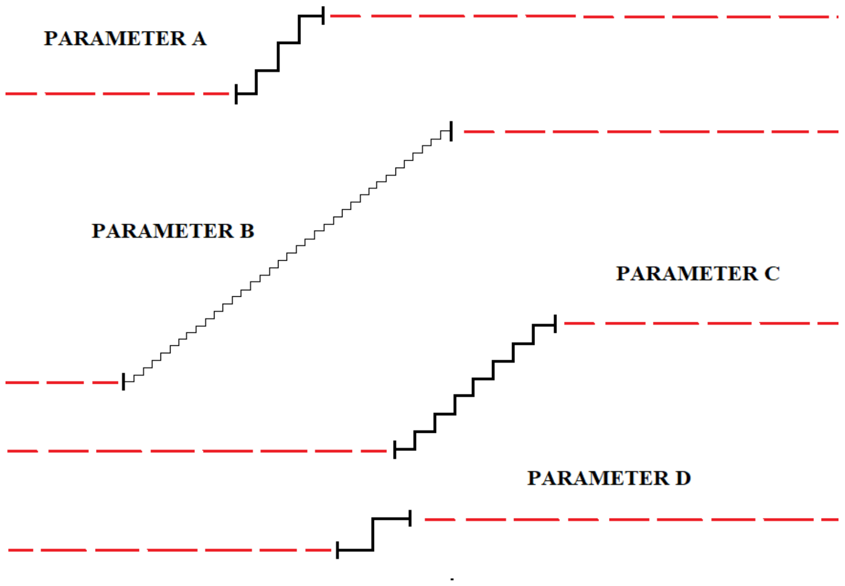


Figure 3.9. Example of an input space for a safety-critical system. The N-dimensional input space is defined by the number of states (x-axis) and the values (y-axis) of each parameter. A safety-critical system is guarded against out-of-range values.

Random (uniform) sampling provides an unbiased alternative to focus on an assumed user profile. This approach is independent of the number of input parameters of the system. Figure 3.9 shows the defined range for parameters

A-D. Within these ranges, samples need to be randomly selected. This will reach all the code that is used during normal operations, as unreachable code is not allowed in this type of software [30]. The remaining code that handles errors need to be covered as well, this is done by selecting input from the dashed area of the parameter ranges. The details, on how and to what extent such values should be sampled are one of the focus areas of the upcoming PhD-part of thesis.

Improving the prediction of the remaining faults does not improve the software. In order to find and remove faults and reach an acceptable level of remaining faults, a large number of tests need to be performed. How many tests that are required depends on a number of factors, e.g., initial amount of faults, distribution of faults, acceptable level of remaining faults. The example from section 3.2 has 100 faults comprising 1% of the input space. As stated, the probability of 5000 tests finding the specific 37 faults, is close to zero. Figure 3.10 shows an analysis of the probability of fault finding as a function of the length of the test suite. This shows the importance of finding ways and methodologies to drastically increase the test coverage.

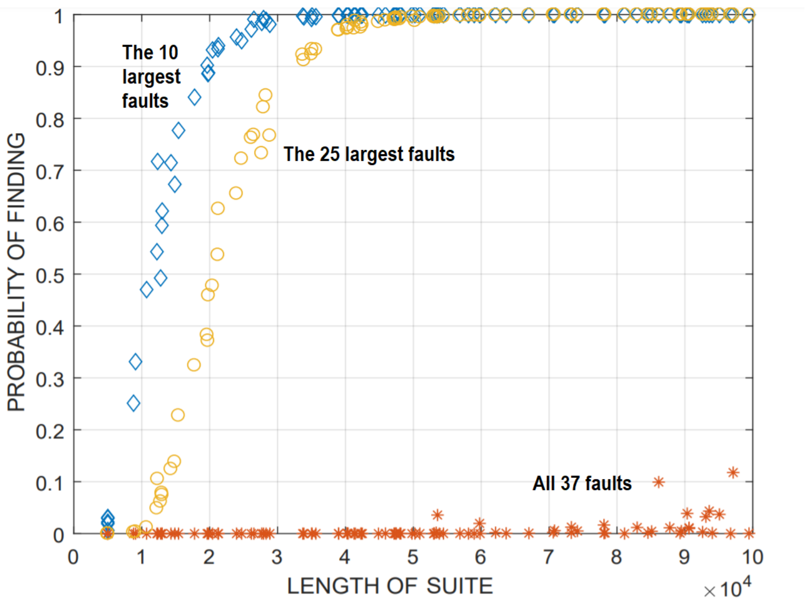


Figure 3.10. Simulation results. Probability of a test suite finding the 10 (blue), 25 (yellow) largest or all 37 (red) triggered faults. This illustrates that a certain test suite length is required to identify a given set of faults. The probability of 5000 tests finding the specific 37 faults is close to zero <=> it is unlikely that there are only 37 faults.

4. Publications Included In the Thesis

Paper II

Prediction of Undetected Faults in Safety-Critical Software

Johan Sundell, Richard Torkar, Kristina Lundqvist, and Håkan Forsberg

Status: Published at 2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)

Abstract

Safety-critical software systems need to meet exceptionally strict standards in terms of dependability. Best practice to achieve this is to follow and develop the software according to domain specific standards. These standards give guidelines on development and testing activities. The challenge is that even if you follow the steps of the appropriate standard you have no quantification of the amount of faults potentially still lingering in the system. This paper presents a way to statistically estimate the amount of undetected faults, based on test results.

Paper III

Safety-Critical Software - Quantification of Test Results

Johan Sundell, Kristina Lundqvist, and Håkan Forsberg

Status: Published at WoSoCer, The 10th IEEE International Workshop on Software Certification, 2020.

Abstract

Safety-critical software systems have traditionally been found in few domains, e.g., aerospace, nuclear and medical. As technology advances and software capability increases, such systems can be found in more and more applications, e.g., self-driving cars, autonomous trains. This development will dramatically increase the operational exposure of such systems. All safety-critical applications need to meet exceptionally stringent criteria in terms of dependability. Proving compliance is a challenge for the industry and there is a lack of accepted methods to determine the status of safety-critical software. The regulatory bodies often require a certain amount of testing to be performed but do not, for software systems, require evidence of a given failure rate. This paper addresses quantification of test results. It examines both theoretical and practical aspects. The contribution of this paper is an equation that estimates *the remaining undetected faults in the software system after testing. The equation considers partial- and limited test coverage. The theoretical results are validated with results from a large industry study (commercial military software). Additionally, the industry results are used to analyze the concept of entropy also known as Shannon information, which is shown to describe the knowledge gained from a test effort.*

4.1 Other Publications

Paper I

Model of Fault Distribution in Complex Safety Critical Systems.
Johan Sundell
Status: Not published. MDH reference: 2018/0071

Abstract
The suggested model is developed for the purpose of investigating the relationship between test coverage and its effect on a given fault distribution in large complex safety critical n-parameter software systems. The faults are represented by subspaces of the entire volume which represents the entire input space of the system. The behavior of the system is considered to be either correct or incorrect. Inside the subspaces the system behaves faulty. The shape of the subspaces have no meaning only the size of its volume. A uniform distribution of test points leads to predictable and quantifiable fault detection.

This early work lays the foundations for Papers II, and III, and many interesting conclusions could be drawn from the described model. However, the model has been evolved since the paper was written. A simpler and more intuitive way of explaining the dynamics would have been used today, i.e. the proportionality between the volumes of the subspaces (representing faults) and their risk of being triggered. The analogy of fault size and risk of being triggered is used by others, and is often easier to understand than multidimensional projections of overlapping subspaces. Moreover, newer versions of the model allows for studying other distributions of fault sizes, e.g. Normal, Lognormal, Pareto, Beta. In fact, this area has turned out to be very interesting from a scientific point of view.

5. Related Work

The problem of predicting the failure rate for safety-critical software has been an issue for decades. The problem has been described by many, e.g., McDermid and Parnas [24, 27]. McDermid talks about a "prediction gap" between the target failure rate and the failure rate that can be proven in testing. Dealing with the strict failure rate requirements was early on seen as very challenging, if not a futile problem. Parnas et al., [24, 27] state "Because of the large number of states (and the lack of regularity in its structure), the number of states that would have to be tested to assure that software is correct is preposterous."

The strict requirements mean that conventional methods and metrics, e.g., reliability growth model, cannot be applied [14, 21, 26].

The problem has been viewed from different angles. The concept of testability [31], was introduced by Voas, and is defined as the probability of faulty software failing. This concept has been further elaborated by Bertolini and Strigini [4, 5, 6]. A lot of focus has been on which conclusions that can be drawn from long test suites with no encountered failures. Voas et.al., describe in [32] how expected failure rate can be calculated based on the number of test cases and how they were sampled (user profiles), as long as no failures are encountered. In [7] Bertolino analyses the two extreme positions of assessment of safety-critical software, i.e. the statistical position, requiring data from realistic testing/operation, versus the probability of the software being fault free (perfect).

Other noteworthy related work includes Arcuri's et al., analysis of how many test cases that are required to cover a given number of targets [3]. Bishop has presented papers about worst case bound and worst case reliability function [8, 9].

Another approach, which aims at describing the state of the system, is to look at the entropy or Shannon information of the software. Clark et al., [11] state about entropy, "It is a negative quantity that measures ignorance - uncertainty about the outcome of the experiment of sampling the random variable". Testing reduces the ignorance of the system's behavior, see [11, 33].

The concept of combinatorial testing is related to the work presented in this thesis. Aspects of this research field help to explain some of the dynamics of fault detection in relation to test coverage and number of parameters [19].

The above related work well describe the problem and the mathematical challenges associated with testing safety-critical software. The work presented in this Licentiate thesis builds on this theoretical foundation but focuses on the gap between theory and practice. The approach presented here, accepts the theoretical limitations previously described, but takes the pragmatic stance of quantifying the remaining faults, and thereby reducing the "prediction gap". The work has a strong industry focus and is aimed at direct implementation in such a setting. The quantification (prediction of undetected faults) is based on parameters known in an industry test process.

6. Conclusions and Future Work

In this thesis, we have investigated the relationship between number of tests, test coverage, and detected/undetected faults for safety-critical software systems. We have reached results in the form of equations describing how the scale of the testing affects the remaining undetected faults in a system. A method has been defined to assess and quantify results from a structured test process.

The model used for this work has been implemented in MATLAB. It has proven to be a useful tool for investigating the dynamics of fault detection as a function of number of test cases.

6.1 Improving the Prediction equation

There are always aspects of the prediction equation that can be modified and improved.

The distribution of the remaining undetected faults changes as more and more faults are discovered, i.e., the bigger faults are more likely to be detected first. At some point the distribution of the remaining faults is no longer Gaussian. This means that at some point (number of test cases), a modified equation may be more accurate.

Another improvement would be an identification of the fault distribution and a subsequent adaptation of the prediction equation. The relationship between the test data υ/τ (unique faults / number of test cases) and ε/τ (triggered faults / number of test cases) can be used as an indicator of the shape of the fault distribution. For example, if $\upsilon(\tau) \approx \varepsilon(\tau)$ the fault distribution is highly granular and may be more uniform. One the other hand, if $\upsilon(\tau) \ll \varepsilon(\tau)$ a few larger faults dominate and the fault distribution may be better described as a Pareto distribution. Bayesian analysis, could provide a flexible and a more accurate approach for analysis of unknown and unusual fault distributions.

Technology is emerging that automatically can determine the root cause of a triggered fault. The rate of discovery of new undetected faults provides useful information about fault distribution in the system. If this information can be exploited the accuracy of the prediction can be assumed to be improved.

The concept of using entropy shows promise. It would be interesting to investigate limit values for acceptance of software, i.e. determine threshold values and what they mean in terms of expected failure rates.

6.2 Methodology for Autonomous testing

This methodology, which is the focus of the full PhD-scope, aims at implementing and optimizing autonomous random testing and at the same time provide a prediction of the remaining faults in the software. The so called "autonomy", i.e., no user interaction is required, allows for continuous testing which subsequently drastically increases the test coverage. The automated generation of input is in accordance with what the prediction equation requires, i.e. sampling from a uniform distribution.

The optimization of such a test tool, though implemented and utilized, is not part this Licentiate thesis.

6.3 Dynamics of software testing using failure density and entropy

The purpose of software testing is to verify that the software does what it is supposed to do, i.e., fulfills the requirements. It also detects bugs, finds performance problems, and reduces development costs by finding problems before it is used. The amount of bugs (total risk of triggering faults) that remains after a test campaign primarily depends on the four factors, listed in section 3.1.3.

When the SUT (System Under Test) reaches the testing phase it has a given number of bugs, i.e., a given fault distribution, granularity (number of faults), and failure density. At this point the quality of the software has been determined by factors such as the experience and qualifications of the staff, type of development process, and environment (programming language, tools etc). The effect of the testing that follows depends on the state of software prior to testing. A holistic approach to improving the quality of safety-critical software should not only focus on the testing. This section discusses some aspects of how these factors interact. The dynamics of the interaction are complex. In general, less bugs (a low failure density) prior to testing is good, and fewer large bugs are better than many small bugs. On the other hand for a given number of bugs the effect of the testing is better if the bugs are bigger. Fault density and entropy can clarify how this works.

The concepts of failure density and entropy are related as shown in Figure 6.1. Both concepts can provide a prediction related to the operational failure rate of a deployed system.

If the expected user profile is indeed uniform sampling of the full input space, the failure density is the predicted operational failure rate. However, if the expected user profile constitutes a subpart of the input space, the failure density is a prediction of the average failure rate for all such subparts. The obvious conclusion is that a reduction of the failure density (and entropy) is always desirable. Detection and subsequent removal of faults can be achieved

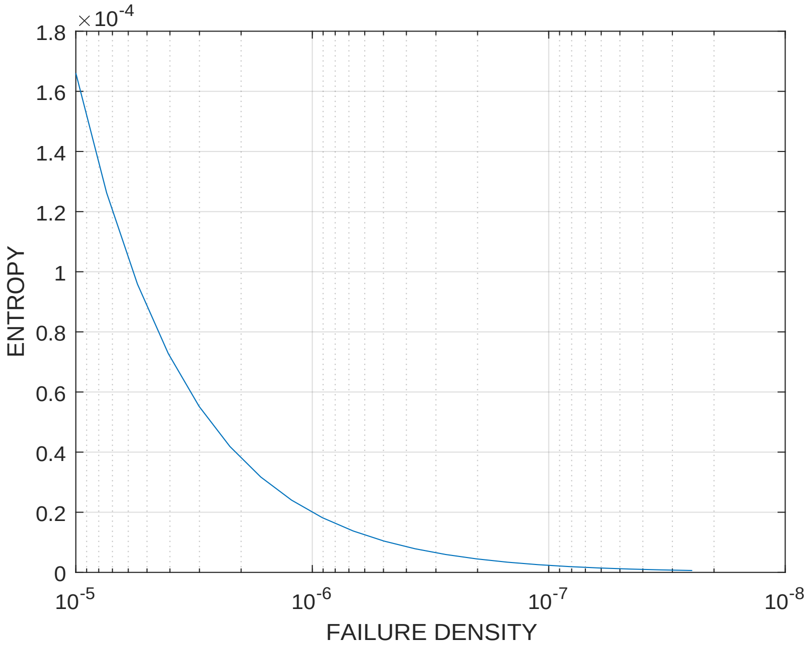


Figure 6.1. Entropy vs failure density. An entropy value 2.3×10^{-6} corresponds to a failure density of 10^{-7}. (Simulated data)

by increasing the length of the test suites. However, an alternative approach is to make the existing faults more detectable. Testing the system will reduce the failure density and entropy. Intuitively one might assume that the initial state of the software is the most important factor in the end state, e.g., software that has less faults will result in a lower post-testing entropy. However, this is not always true in these extreme cases. Figure 6.2 shows how perceived entropy (calculated from available parameters τ, ε, υ and sp), changes for long test suites with 500k tests.

Three different scenarios were run four times each. All had 158 faults, but with different initial failure densities. The upper starts at a failure density of 0.005 giving an entropy of 0.0382. The lower two starts at 0.001/0.01 and 0.0001/0.0013 respectively. Here the final entropy for the lower two is approximately 0.001 and for the upper 0.0003, which is roughly 3 times lower. In this case, starting with a higher entropy, means ending up with better final state. The reason for this paradox is that the size of the faults in the upper curve are bigger, and thus easier to find. From a testing point of view, minimizing the risk of triggering each fault might not be the optimal approach. This aspect is worth noticing for this type of software.

The dynamics of the problem are complex, Figure 6.3 shows that for a given failure density, the number of faults triggered (ε) is more or less constant but

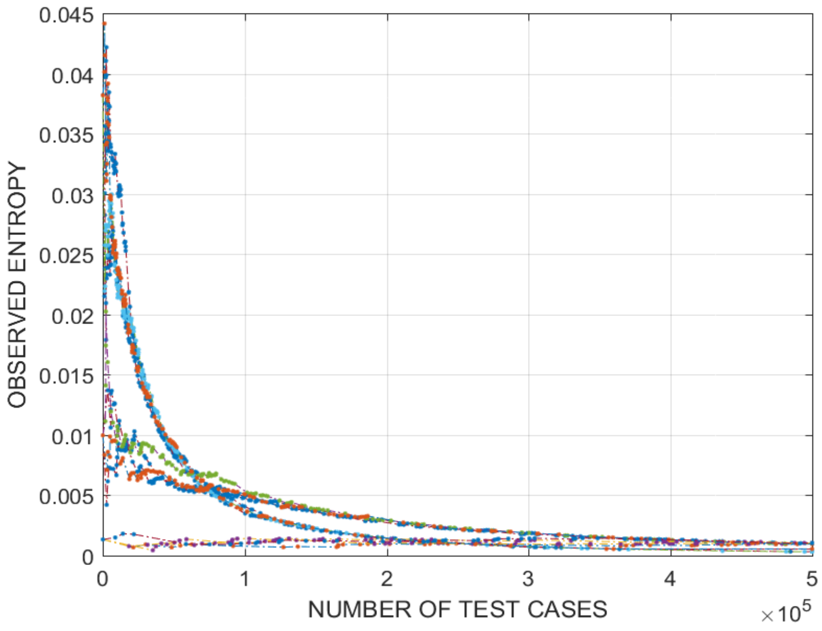


Figure 6.2. Observed entropy vs number of test cases. The full input space is sampled. Three scenarios starting at 0.0382, 0.01 and 0.0013. Each scenario is run four times each. The scenario starting highest (0.0382) ends up lowest after 500k tests. (Simulated data)

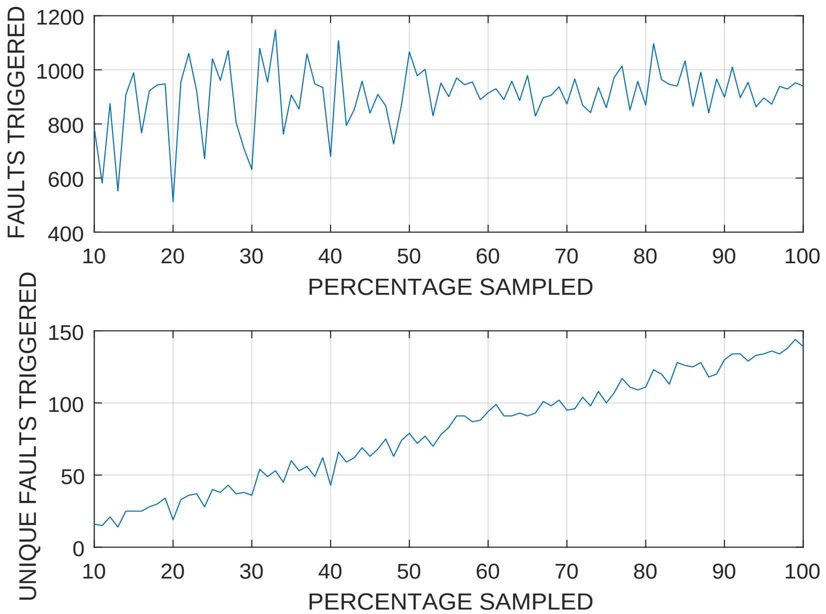


Figure 6.3. Triggered faults vs percentage covered. For a given initial failure density, here 0.01 and 158 faults, and 100k tests, the total number of triggered faults are constant and the number of unique faults triggered is proportional to the percentage sampled. (Simulated data)

the number of unique faults triggered increases linearly with the test coverage. This is expected and can be explained by an example, e.g., if you focus testing on one half of the input space (50%) you will trigger faults as many times (ε) as if you were testing the full input space. However, you are triggering the same faults over and over, as you can only find and trigger faults in that half, i.e., the number of unique faults (υ) is lower.

Another way of illustrating the dynamics, and the importance of test coverage is shown in Figure 6.4. Here it is shown that the final entropy ($\mathscr{H}$) is linearly correlated to how much of the input space that is covered.

As can be seen in Figure 6.2, the change in entropy tapers off with more and more tests, as there are less faults to be found. Both a stabilized $\mathscr{H}$ and $\Delta\mathscr{H}$ could be used as a stopping criteria for testing. A $\Delta\mathscr{H}$ value close to 0 means that there is little effect of further testing.

Future work is needed to find ways to make faults more detectable. Additionally, work is needed to define stopping criteria for when testing can be stopped, i.e., define acceptable levels for failure density/entropy.

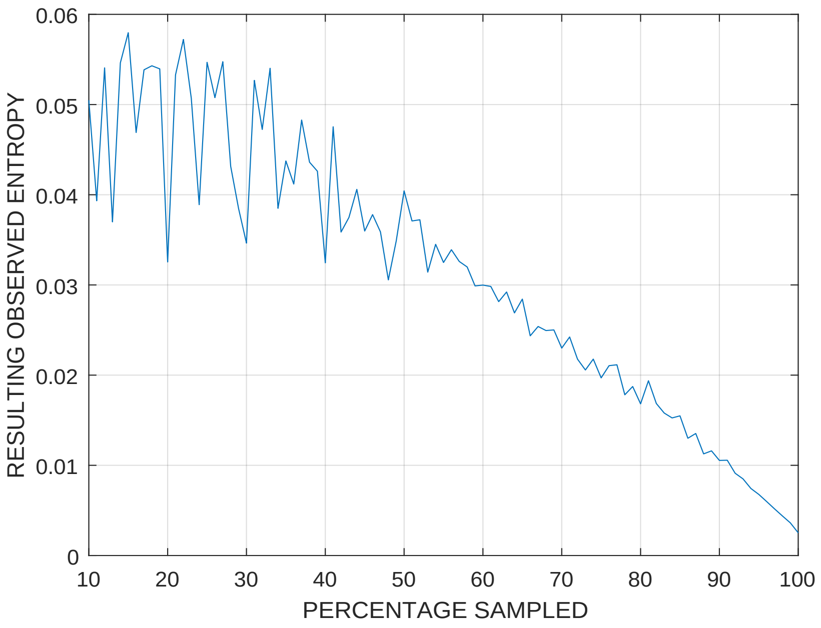


Figure 6.4. Entropy vs percentage covered. For a given initial failure density, here 0.01 and 158 faults, and 100k tests, the final entropy is reduced as the percentage sampled is increased. (Simulated data)

7. Thesis Outline

The proposed title of this thesis is Safety Critical Software - Test Coverage vs Remaining Faults. The thesis will be written as a collection of papers. Besides these papers, the thesis will also include a summarising chapter of the compilation dissertation. The outline of the thesis is as follows.

Part I: Thesis

1 Introduction
Introduction to the research problem and objective of the thesis.

1.1 Problem Statement
The problem and its importance is presented.

1.2 Research Objectives
Description of the full PhD scope and how this Licentiate part fits in.

1.3 Thesis Objectives
Description of the Licentiate objectives.

2 Thesis Contributions
The contributions of the Licentiate are listed in detail.

2.1 Conventional Approach
As in this proposal.

2.2 Alternative Approach

3 Research Process
Overview of the included papers and their contribution.

4 Publications Included In the Thesis

4.1 Paper II: Prediction of Undetected Faults in Safety-Critical Software

4.1 Paper III: Safety-Critical Software - Quantification of Test Results

5 Related Work
Overview of important and inspiring work that forms a foundation to the thesis.

6 Conclusions and Future Work

6.1 Improving the Prediction equation

6.1 Methodology for Autonomous testing

8 Thesis Outline

Part II: Bibliography

References

[1] Knight Capital filings show scant board duty for tech risk. *Reuters*, August 2012.

[2] NATIONAL HIGHWAY TRAFFIC SAFETY ADMINISTRATION. Distracted driving, 2021.

[3] A. Arcuri, M. Z. Iqbal, and L. Briand. Random testing: Theoretical results and practical implications. *IEEE Transactions on Software Engineering*, 38(2):258–277, March 2012.

[4] Antonia Bertolino and Lorenzo Strigini. Using testability measures for dependability assessment. In *Proceedings of the 17th International Conference on Software Engineering*, ICSE 95, pages 61–70, New York, NY, USA, 1995. Association for Computing Machinery.

[5] Antonia Bertolino and Lorenzo Strigini. On the use of testability measures for dependability assessment. *Software Engineering, IEEE Transactions on*, 22:97 – 108, 03 1996.

[6] Antonia Bertolino and Lorenzo Strigini. Acceptance criteria for critical software based on testability estimates and test results. In Erwin Schoitsch, editor, *Safe Comp 96*, pages 83–94, London, 1997. Springer London.

[7] Antonia Bertolino and Lorenzo Strigini. Assessing the risk due to software faults: Estimates of failure rate versus evidence of perfection. *Softw. Test., Verif. Reliab.*, 8(3):155–166, 1998.

[8] P. G. Bishop and R. E. Bloomfield. A conservative theory for long term reliability growth prediction. In *Proceedings of ISSRE 96: 7th International Symposium on Software Reliability Engineering*, pages 308–317, Oct 1996.

[9] Peter G. Bishop and Robin E. Bloomfield. Worst case reliability prediction based on a prior estimate of residual defects. In *Proceedings of the 13th International Symposium on Software Reliability Engineering*, ISSRE 02, page 295, USA, 2002. IEEE Computer Society.

[10] R.W. Butler and G.B. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19(1):3–12, 1993.

[11] David Clark, Robert Feldt, Simon Poulding, and Shin Yoo. Information transformation: An underpinning theory for software engineering. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 599–602, Piscataway, NJ, USA, 2015. IEEE Press.

[12] Dijkstra EW. The humble programmer. *Commun*, 10(EWD 340):859–866, 1972.

[13] A. German and G. Mooney. Air vehicle software static code analysis lessons learnt. *Aspects of Safety Management, F. Redmill and T. Anderson*, pages 175–193, 2001.

[14] D. Hamlet and J. Voas. Faults on its sleeve: Amplifying software reliability testing. In *Proceedings of the 1993 ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '93, pages 89–98, New York, NY, USA, 1993. ACM.

[15] H. J. Holz, A. Applin, B. Haberman, D. Joyce, H. Purchase, and C. Reed. Research methods in computing: what are they, and how should we teach them? *SIGCSE Bulletin*, 38(4):96–114, 2006.

[16] IATA. Iata controlled flight into terrain accident analysis report 2008-2017 data. Standard ISBN 978-92-9229-717-6, IATA, Montreal, Geneva, 2018.

[17] International Electrotechnical Comission. Nuclear power plants—Instrumentation and control important to safety—General requirements for systems. Standard IEC 61513:2011, International Electrotechnical Comission, Geneva, Switzerland, aug 2011.

[18] ISO Central Secretary. Road vehicles—Functional safety. Standard ISO 26262-1:2011, International Organization for Standardization, Geneva, Switzerland, nov 2011.

[19] D. Richard Kuhn, Dolores R. Wallace, and Albert M. Gallo. Software fault interactions and implications for software testing. *IEEE Trans. Software Eng.*, 30(6):418–421, 2004.

[20] Nancy Leveson. Medical devices: The therac-25.

[21] B. Littlewood and L. Strigini. Validation of ultrahigh dependability for software-based systems. *Commun. ACM*, 36(11):69–80, November 1993.

[22] B. Littlewood and L. Strigini. Validation of ultra-high dependabilit - 20 years on. 2011.

[23] J. L. Lyons and Yvan Choquer. Ariane 5 : Flight 501 failure report by the inquiry board paris , 19 july 1996 the chairman of the board : Prof. 2013.

[24] J. A. McDermid and T. P. Kelly. Software in safety critical systems: Achievement and prediction. *Nuclear Future*, 2(3):140–146, 2006.

[25] Richard McElreath. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan, 2nd Edition*. CRC Press, 2 edition, 2020.

[26] D. R. Miller. Making statistical inferences about software reliability. Technical Report NASA CR-4197, National Aeronautics and Space Administration, Scientific and Technical Information Division, 1988.

[27] D. L. Parnas, A. J. van Schouwen, and S. P. Kwan. Evaluation of safety-critical software. *Commun. ACM*, 33(6):636–648, June 1990.

[28] Special C. RTCA. *DO-178C, Software Considerations in Airborne Systems and Equipment Certification*. Special Committee 205 of RTCA, 2011.

[29] Johan Sundell. Model of fault distribution in complex safety critical systems. Standard 2018:0071, Mälardalen University - MDH, Västerås, Sweden, 2018.

[30] Johan Sundell, Richard Torkar, Kristina Lundqvist, and Håkan Forsberg. Prediction of undetected faults in safety-critical software. In *2nd IEEE Workshop on NEXt level of Test Automation*, April 2019.

[31] Jeffrey Voas and Keith W. Miller. Software testability: The new verification. *IEEE Software*, 12:17–28, 1995.

[32] Jeffrey M. Voas, Christoph C. Michael, and Keith W. Miller. Confidently assessing a zero probability of software failure. In Janusz Górski, editor, *SAFECOMP '93*, pages 197–206, London, 1993. Springer London.

[33] Linmin Yang. *Entropy and Software Systems: Towards an Information-theoretic Foundation of Software Testing*. PhD thesis, Pullman, WA, USA, 2011. AAI3460455.