Safety-Critical Software - Quantification of Test Results

Johan Sundell, Kristina Lundqvist, Håkan Forsberg
IDT - School of Innovation, Design and Engineering
Mälardalen University
Västerås, Sweden
johan.sundell@mdh.se, 0000-0003-0904-3712, 0000-0002-0933-6059

***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 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.

***Index Terms*—safety-critical, failure rate, prediction, software, entropy, test**

I. Introduction

The technological advances allow for an increasing number of safety-critical applications. More and more complex descisions are handed over to software based systems. In the past such safety-critical systems have been found in a few domains such as aerospace and nuclear. 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 [1] or IEC 61513 [2], have evolved that gives guidance on best practice. However, there has been the lingering problem of actually proving the target failure rates prior to deploying the systems. McDermid [3] 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. The consensus in the safety-critical community is that a fault density of 1 per kLoC (1000 lines of code) is considered to be world class [3], i.e., a safety-critical program with 100 kLoC can be expected to have approximately 100 faults.

Despite that a lot of theoretical work have been done, for safety-critical software there is a lack of methods and strategies to deal with the issue of predicting the failure rate or, with confidence, determine the fault density of the software. In [4], Littlewood and Strigini describes the slow progress in this area since the early nineties. The drive towards autonomous vehicles, advanced IoT, artificial intelligence etc, means that this problem is further aggravated. More and more safety-critical systems will be in use in the future. Uncertain predictions of their failure rate, risk coming with a high pricetag, as such systems become more exposed [5].

The contribution of this paper is to present a method for determining the status of safety-critical software that is applicable to industrial sized real software. This method consists of taking a sufficiently large sample and use the results to statistically draw conclusions about what remains undetected. The equation from Sundell et al. [6], that estimates the undetected faults when the full input space is sampled, is generalised to consider partial test coverage. The input to this equation are parameters known in an industry setting, after a test effort. The results are validated with a real industry case study.

This paper is organized as follows, section II covers related work, that is in section III followed by a theoretical presentation of extrapolation of partial sampling. The case study is presented in section IV, which is followed by a section on entropy. Section VI summarizes the results. The paper ends with two sections on further analysis of the results and drawn conclusions.

II. 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 [3], [7]. The strict requirements mean that conventional methods and metrics, e.g., reliability growth model, cannot be applied [8]–[10]. The problem has been viewed from different angles. The concept of testability [11], was introduced by Voas, and is defined as the probability of faulty software failing. This concept has been further elaborated by Bertolini and Strigini [12]–[14]. A lot of research has been focused on which conclusions can be drawn from long test suites with no encountered failures. Voas et al. describe in [15] how expected

978-1-7281-7735-9/20/$31.00 ©2020 IEEE
DOI 10.1109/ISSREW51248.2020.00089

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 [5] Bertolino and Strigini analyze 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 [16]. Bishop and Bloomfield have presented papers about worst case bound and worst case reliability function [17], [18].

The work presented in this paper is related to above work but is focusing more on practical industry implementation. Much of the work in this paper is based on attempting to implement and adapt the results reported by Sundell et al. [6], which presents an equation for prediction of undetected faults. This equation anticipates that the software is indeed faulty and that all the faults are not detected during testing.

A related approach, which aims at describing the state of the system, is to look at the entropy or Shannon information. Clark states in [19] about entropy, "It is a negative quantity that measures ignorance – uncertainty about the outcome of the experiment of sampling the random variable". In this paper, the outcome is interpreted as being, the risk of triggering a fault when the software is executed. This defines the ignorance of the outcome when the software is executed.

III. Theory - Extrapolation of partial sampling

The equation presented in [6] predicts the remaining faults when the full input space is sampled. The equation is based on the predictibility you get from uniform sampling. This section analyzes the cases when only a limited part of the input space can be reached. As discussed in subsection IV-B, achieving a high test coverage can require a lot of effort. However, a test suite that targets only a subspace of the input space can still be used to draw conclusions of the remaining faults in the software. This assumes that the untested parts of the software is of the same quality as the tested parts, in terms of fault density.

A. Assumptions

The theoretical part of this work is based on the same assumptions stated in [6], with one exception. The area of interest here is the situation where the full input space, for whatever reason, can not be reached and thus assumption 1 is removed, see below. Instead we add an assumption 7, that the (reachable) sample is assumed to be representative of the system in terms of present faults.

Assumptions:

1) ~~The input and state space of the system is reachable.~~
2) All parameters are defined at runtime.
3) The system has a deterministic behavior.
4) Uniform sampling of the input space is used.
5) More than 1,000 test cases are generated.
6) More than 50 faults are present in the system prior to testing.

Additional assumption:

7) The tested part is representative in terms of fault density.

B. Rationale

The assumptions refer to the context of medium to large safety-critical software systems and their rationale are explained in more detail by Sundell et al. [6]. The rationale can be summarized in the following way.

The development standards for safety-critical software, e.g., DO-178C [1], IEC 61513 [2], constrain the resulting software to have unique characteristics, different from software in general. There are a number of guidelines and recommended practices that aim to make the software deterministic and reliable. There are restrictions on which program languages that can be used, dynamic memory allocation, initialization of parameters etc. Moreover, the standards typically require removal of dead- and unreachable code. If you can not test a part of the code it needs to be removed. Thus, the final code is reachable.

Despite all the precautions, a fault density of 1 (safety-critical) fault/1kLoC is considered to be world class [3].

The other assumptions describe how the random testing is performed.

Figure 1, shows how a test suite in accordance with the assumptions can be modeled in 3D.

The model corresponds with what Littlewood and Strigini proposes, in [9] they state; *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 fault size concept is used in this paper. A large fault means that a relatively large partition of the input space can trigger a specific fault. This can also be expressed as, and is equivalent to, occuring at a large rate during random sampling.

C. Proposed equation

Equation 1, described by Sundell et al. [6], requires a random sampling of the full input space. The failure density is defined as the risk of a random input vector triggering a fault. If random input is the user profile, the failure density is equal to failure rate.

$$\begin{aligned}
&\text{Number of tests} = \tau \\
&\text{Number of failures detected} = \epsilon \\
&\text{Number of identified unique faults} = \upsilon \\
&\text{Failure density} = \rho \\
&\text{(sum of remaining undetected faults)} \\
&\lambda = \upsilon^2/(\epsilon \times \tau) \\
&\rho = \lambda \times e^{-\lambda}
\end{aligned} \tag{1}$$

When a subspace is sampled the above equation is valid only for the sampled/tested part. The untested part can be predicted by ϵ/τ. The Central Limit Theorem ensures that the ϵ/τ-term

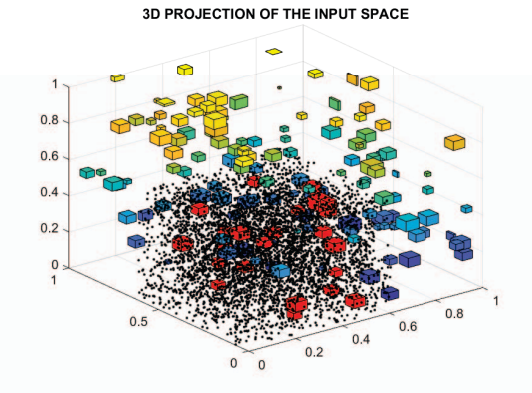

Fig. 1. Test suite sampling a subspace. The full volume represents a 3D projection of the entire input space. The boxes represent input that trigger faults, their volumes are proportional to their respective risk of being triggered. Points are samples drawn in the test suite. The axes span a projection of the normalized input space.

converges to a prediction of the initial failure density of the tested part [20]. By summing the prediction for the tested- and untested parts the following Equation 2 is derived:

$$\begin{aligned} &\text{Percentage of input space sampled} = sp \\ &\lambda = v^2/(\epsilon \times \tau) \\ &\rho = sp \times \lambda \times e^{-\lambda} + (1 - sp) \times (\epsilon/\tau) \end{aligned} \tag{2}$$

The theoretical results, presented in section VI, show that partial sampling can be used to provide an acceptable estimate of the undetected faults. Sampling as little as 25% - 30% of the input space will give the same correlation as a full sampling strategy, and an acceptable low mean error, see Table I and Figure 5.

IV. Case Study - Test Coverage from Random Input

The industrial case study required implementation of random testing. The focus was to study which test coverage random testing could achieve with complex commercial software, i.e., investigate how well the model, presented by Sundell et al. [6], holds up in an industry setting. This model makes the idealisation, that a certain input generates a certain output, this is often an inadequate model of real world systems. Complex systems can, at runtime, be in many different states, which can yield different results.

In this study, the SUT (System Under Test) processes the history of parameters, i.e., the output depends on previous input, then a random generation of input has to take this in to consideration (grey box approach). As shown in this paper, such a tailoring of the input generation, allow for a greatly increased test coverage. It is also shown that the test coverage is very important for maximizing the knowledge of the system, during testing. The case study was performed in collaboration with the industry, and in parallel with the existing development process of the industry project. All findings were reported to the industry project management. The SUT is a subsystem that is used by several military platforms, including fighter jets. The software is classified as being of low/medium criticality.

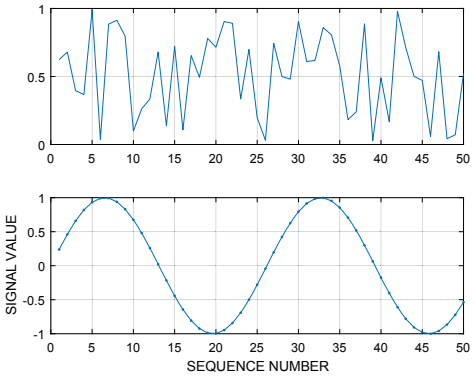

Fig. 2. A randomly generated signal (top) vs a sine curve (bottom). The chance of randomly generating a specific type of curve, e.g., sine, is low.

A. System Under Test - Description

The company's commercial presentation of the system:

"Self-defence for airborne platforms means knowing whether anyone is observing me or making me a target. That requires keeping track of every type of signal out there. This EW (Electronic Warfare) system is designed to provide self-defence in sophisticated, diverse and dense threat environments."

Context, as defined by Jedlitschka et al. [21]:

* application type (EW - Electronic Warfare),
* application domain, (Aerospace),
* type of company (Medium sized),
* experience of the participants (Professionals),
* time constraints (Commercial),
* process (Standardized),
* tools (Commercial),
* size of project ($>$ 1k person months)

B. Black box vs grey box

The initial test suites were performed without considering the structure of the system software, a.k.a. black box testing. Each stream of input was randomly generated and executed. The function test coverage, that was reached for module A (a main software module containing the signal shape logic), was 42%.

An analysis of the system software revealed that it is branched based on the shape of the incoming sequences of a few parameters. The software is identifying signals as being sine-, triangular-, random shaped, etc. The reason for the rather low coverage is that the uniform random generation of a signal is unlikely to generate a certain shape in finite time, see Figure 2.

When the generation of input was modified to include signal streams in the shapes that the software scans for, the test coverage was significantly improved. This consideration of the structure of the software is called grey box testing. For module A, a 96% function- and a 72% condition/decision coverage was reached. Analysis of the missing 4% of the

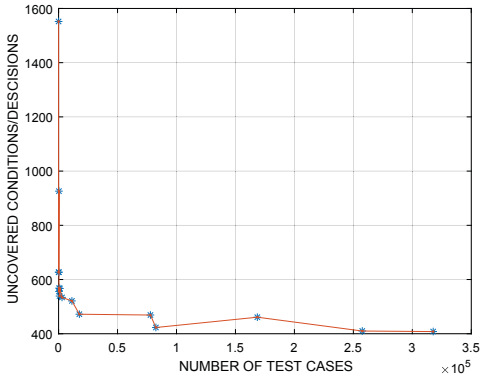

Fig. 3. Uncovered condition/decisions vs number of tests. The values are taken from independent test suites using the same generation of input. The curve converges to a quasi stable level.

functions, in module A, shows an interlinked set of functions that were never called. Despite that more than 3 million random simulations were performed. The top function in the missing 4% - call hierarchy were found to have an if-statement that never evaluates to TRUE. The if-statement had a comment stating *"This should not happen"*. Such hard to reach functions can be considered to be candidates for removal, but can be tested by having a tool or script directly call them. However, if they cannot be triggered on a system level - their system effect cannot be tested and evaluated.

An observation was made that, both the function- and condition/decision coverage stays, more or less, asymptotically stable due to the statistical hurdles described above, see Figure 3.

C. Partial code coverage vs partial input space

During the study we found that, when the input generation was modified it resulted in a corresponding test coverage, independently of the length of the test suite. The data shows that, there is no correlation (0.013) between condition/decision coverage and the length of the test suite.

This means that a given way of generating test input corresponds, and rapidly converges, to a given test coverage, see Figure 3. At this stage, an hypothesis was formulated that a code coverage corresponds to a subpart of the input space, and could be used as an approximation of the *sp*-parameter (percentage of the input space) in Equation 2.

In order to investigate this hypothesis, a number of test suites were performed, where both the code coverage and the length of the test suites were varied. All of these test suites were performed using the same development build or version of the software. The code coverage was determined using the tool BullseyeCoverage [22]. This tool can determine if a function, in the code, has been called (function coverage) and if a conditional statement has been evaluated (condition/decision coverage). Ideally, the results from the proposed Equation 2 should be compared with a complete list of faults with their respective probability of occurence. However, this would require exhaustive testing of the software and such an effort is infeasible for software of this size.

Instead, the validation was done against one parameter that remains constant, i.e., the initial failure density (sum of faults). The real value of this parameter is constant for all test suites, as the same version of the software was used throughout. The initial sum of faults, the initial failure density, can be calculated by adding the detected and undetected faults.

Statistical theory tells us, that the predictions of this sum can be expected to be an approximately normally distributed parameter.

The test coverage, which turned out to be the hardest aspect to control, was varied by commenting out test code that generates certain parts of the test input generation.

The estimated initial failure densities as well as the estimated sum of detected- and undetected faults, are presented in section VI, below. The calculated values show consistency and appear to be a good appoximation to the actual values.

The calculated undetected failure density can be used as a quantification of the current status of the software. Additionaly, the result can be used for calculation of entropy, which is an alternative quantification metric.

V. Information Theory and Entropy

It has been suggested that Shannon information or entropy could be used for analysis of such areas as software testing [19], [23]. The entropy can be interpreted as a measure of ignorance of the outcome of sampling a random variable. Here a lower entropy means less- and a higher means more ignorance.

In this context, the execution of a (safety-critical) system is seen as either having a correct or an incorrect outcome. During the process of testing a system more knowledge is gained, hence the ignorance and entropy is reduced. By calculating the initial- and final entropy states the reduction can be seen as an effect of testing.

The definition of Shannon information is:

$$\mathcal{H} = -\sum_{x \in X} p(x) log_2(p(x))$$

Here p represents the risk of drawing a random set of input that triggers faults. The test effect can be calculated from:

$$\Delta\mathcal{H} = \mathcal{H}_{\text{Initial}} - \mathcal{H}_{\text{Final}}$$

The input to Equation 2 allows for calculation of p, for both the inital state (prior to testing) and the final state. The initial p is given by ϵ/τ, see [20], and the final p is the calculated ρ (remaining undetected faults), from Equation 2.

The results from the simulation study allows for a theoretical analysis of how the test effect (entropy drop) is affected by the length of the test suite and the test coverage. The mean values are summarized in Figure 4.

Furthermore, the analysis can be extended to the data obtained from the industry project. This data was used to calculate the entropy reduction in the same way, as a function of test coverage and length of the test suite. The industry

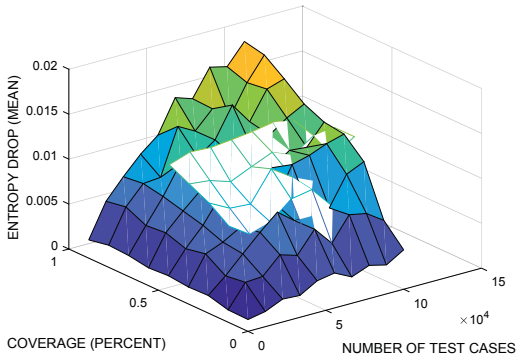

Fig. 4. Test effect. Observed entropy drop (normalized) vs the simulation data. The industry data is superimposed and shown in white. (Simulated- and Industry data)

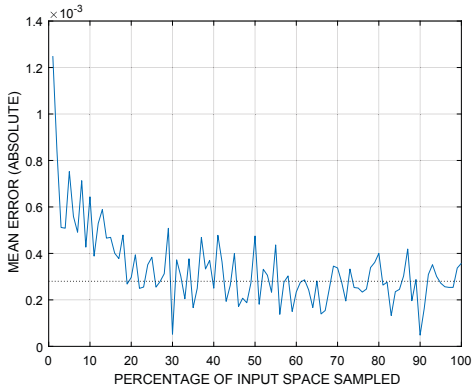

Fig. 5. Mean absolute error vs sampled percentage of the input space. The mean value for each percent is shown. The dotted line depicts the mean error for a full 100% sampling, taken from [6] (Simulated data)

results have been superimposed on to the simulated data, and is shown in white. The industry data (N = 85) covers a subset of the theoretical data (N = 10023).

VI. Results

The study of commercial military software is complicated by the fact that results can be considered commercially sensitive and in some cases classified. In order to mitigate such concerns all the results presented, in this paper, have been "normalized" to the level of 0.01. This is the mean value of the initial sum of faults in the theoretical simulation study. This number shall not be considered to represent the values obtained for the software in the case study.

The values themselves have no relevance in the context that they are presented. Here it is only the relationship between the parameters that are of interest, e.g., the consistency and correlation of the sums of initial faults.

A. Simulation study

The simulation study was performed in the same way, using the same model, as Sundell et al. [6]. The only difference is that here only a part of the projected 3D input space is sampled, see Figure 1. The overall correlation is comparable to previously reported results. However, both correlation and accuracy of the prediction depends on how much of the input space is sampled, see Figure 5 and Table I.

It has been suggested that a lognormal distribution is a better representation of fault sizes [24], [25]. A smaller study (N=595) shows results comparable to the ones presented, correlation 0.99 and mean difference -3.8×10^{-4}. Equations 1 and 2 are not derived from or depend on, fault sizes being normally distributed.

An alternative presentation of the error in the prediction calculation is a histogram, see Figure 7. The peak is on the negative side, e.g., the prediction is, on average, an underestimation.

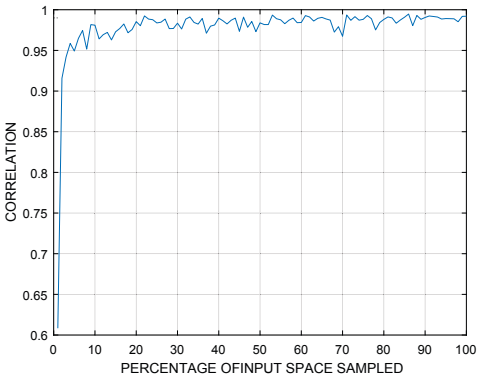

Fig. 6. Correlation between estimate and result vs part of input space sampled. (Simulated data)

B. Case study

The SUT used in the case study is the EW-system described above, in section IV-A. 85 test suites, using random simulations, were performed. The number of tests in the different test suites varied between 2k and 122k. For each test suite the generation of input was modified, in order to vary the code coverage. The resulting condition/decision coverage was between 9 and 61%.

Figures 8, 9 and 10 show different ways to summarize the prediction of detected and undetected faults. The results comprise all the test suites, including the ones that have been

TABLE I
Comparison between predictions and results (simulations). All test suites have a length between 1000 and 100.000 samples.

Nr of Test suites	Corr Coeff	Mean error	Std Dev	Percentage sampled
10023	0.97	-3.3×10^{-4}	1.1×10^{-3}	$0 \rightarrow 100\%$
2481	0.95	-5.0×10^{-4}	1.7×10^{-3}	$0 \rightarrow 25\%$
7542	0.99	-2.8×10^{-4}	7.9×10^{-4}	$25 \rightarrow 100\%$
10024	0.99	-2.8×10^{-4}	2.5×10^{-5}	100%

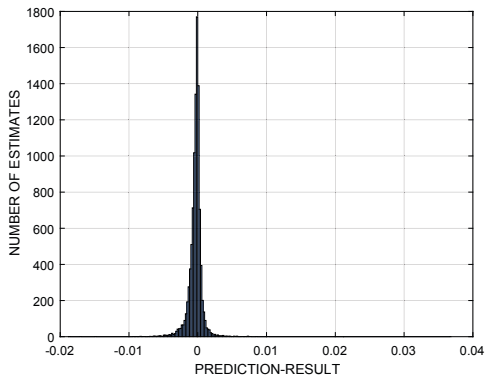

Fig. 7. Histogram of (prediction - result). 10023 test suites. (Simulated data)

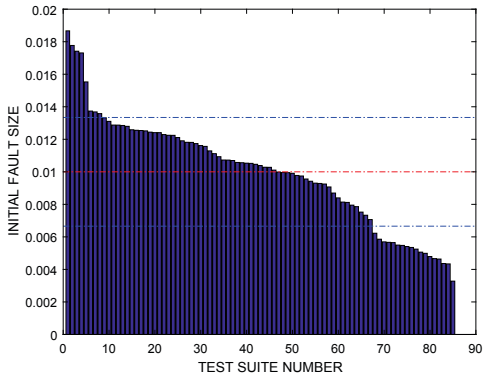

Fig. 8. Estimation of initial sum of faults. The mean is shown by the dashed center line and ± 1 standard deviation by the top- and bottom dashed lines. (Industry data)

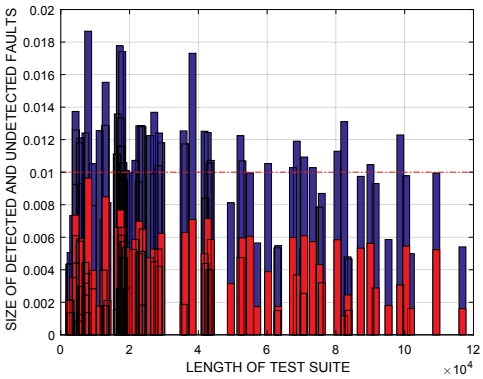

Fig. 9. Detected and undetected faults vs number of test cases (samples drawn). The undetected faults are shown as the taller blue part of the bars. The red bars show the size of the detected faults. The percentage of the input space that is sampled is varied. (Industry data)

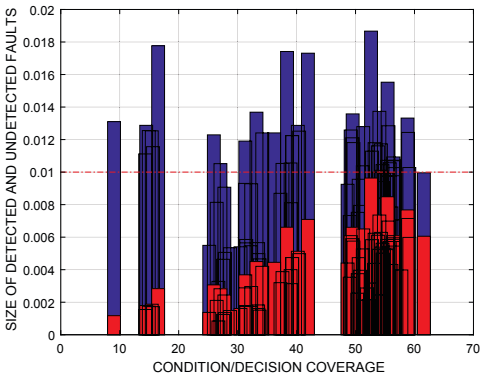

Fig. 10. Detected and undetected faults vs condition/decision coverage (in %). The undetected faults are shown as the taller blue part of the bars. The red bars are the size of the detected faults. The number of test cases are varied. (Industry data)

shown to have a lower correlation, i.e., test suites with $< 25\%$ test coverage.

The main point here is how much the calculated initial sum of faults (detected + undetected) varies from the level 0.01. This level, which is the (normalized) mean, represents the presumed true value. The true value is the same for all test suites as the version of the SUT is the same.

The standard deviation of this data set is 0.003, i.e., $\pm 30\%$ from the normalized level 0.01.

VII. DISCUSSION

This section discusses the findings from the work. This paper looks at interpretation of test results for safety-critical software. This is of additional importance for software which is required to have extremly low failure rates. This paper shows that a modified Equation 1, from [6], can be applied to a limited test coverage. The modified Equation 2, estimates the sum of the remaining undetected faults, based on the partial test coverage.

It is also shown that a random sampling of input can be insufficient to reach an acceptable level of test coverage. In such cases, a grey-box approach might be required. This means that the structure of the software is considered, and the input generation modified, in order to better cover the code.

Results indicate that code coverage data can be used as an approximation to limitations of input space sampling, in this context. Data from the industry show good correlation for both calculation of failure density and entropy.

Futher studies are planned, using safety-critical software in other domains, to validate the results presented here.

A. Strategies for generating random input to maximize test coverage

Today many safety-critical systems are used to monitor, analyze and respond to a large number of input parameters, often in real time. Such systems have to correctly interpret the data and often the history of the data, in order to make predictions and calculate a response.

For example, a sense and avoid system that is used by UAVs (Unmanned Aerial Vehicles) to maintain separation to other flying objects, need to know if an object is approaching or

not. Proximity is not by itself sufficient to trigger a response, as the object might be moving away. More parameters need to be looked at, this means that random generation of the test cases need to reflect this complexity.

Ideally, a random generation of input should cover all code, including the code handling erroneous input. Random testing will then continuously explore more and more of the state space of the system. This ideal setup, allows for an efficient scaling of the test effort, i.e., a network of test resources can then be utilized to significantly improve the overall testing performed prior to deployment. The progress of the testing can be calculated using Equation 2.

B. Prediction equation for a limited test coverage

The theoretical results indicate that partial sampling can be used to provide an acceptable estimate of the undetected faults. Provided that the samples are representative (assumption 7), sampling of as little as 25% will give the same correlation and an acceptable low mean error as the original full sampling strategy, see Table I and Figures 5 and 6.

The practical part of this work supports these results. Equation 1 calculates an estimate of the sum of undetected faults in the software, based on the test results. If only a subset of the software is sampled, defined as either part of the input space or the (pseudo stable) test coverage data, Equation 2 applies.

The structure of Equation 2, is to use Equation 1 on the tested subpart and to add the $epsilon/tau$-term for the untested part. This is then used as an approximation of the untested part. The proportions between the tested/untested parts are of importance. The bigger the tested part is the better, i.e., predicting the last 10% from the result of 90% is more accurate than vice versa.

The additional assumption 7, that the sampled part shall be representative of the system is needed for the extrapolation part of the equation. If one argues that the software has an even ”quality” in terms of fault density, i.e., the faults are occuring throughout the software, then the undected fault density can be calculated by a modified Equation 2. Here the number of test cases are concentrated to a part of the input space, and Equation 1 is only valid for this part. In the remaining part of the input space all the faults are undetected but can be quantified by $epsilon/tau$, i.e., detected failures/number of tests.

There are evidence suggesting that faults are distributed according to the Pareto principle [26], [27], i.e., 80% of the faults are found in 20% of the software. Such an uneven distribution of faults, increases the risk of testing an area that is not representative for the system as a whole. Especially if the sampled part is very small.

However, these studies were not performed on safety-critical software. It is assumed that safety-critical software is more even in terms of quality and fault distribution. The development standards forces, especially for the higher levels (DAL A etc), a substantial analysis effort before code can be commited.

The purpose of, and motivation for the standards are to achieve a low failure rate of the system, by reducing the amount of faults in the software. There are different levels of procedural strictness depending on what level of confidence that needs to be achieved. These levels have different names in the different standards, e.g., SIL (Software Integrity Level in IEC-61508), ASIL (Automotive Safety Integrity Level in ISO 26262) or DAL (Design Assurance Level in DO-178). What they all have in common is the increased rigor and formality that is required during the software development. More and more requirements are added for each criticality level in terms of required test coverage, reviews, organization etc. For higher levels of criticality it is required to review and verify with independence. Which in DO-178C [1] is defined as *Separation of responsibilities which ensures the accomplishment of objective evaluation*, i.e., the developer or the development team can not review themselves. The end result, proven by low failure rates in the safety-critical domains, is an even software quality where the vast majority of faults have been detected and eliminated.

C. Prediction equation applied to industry test results

Equation 2 works in theory but can it be applied and validated against industry data?

Without the possibility of performing exhaustive testing, and thereby completely map out the input- and state space of the system, an alternative strategy was required.

In this case, it was choosen to compare what remains constant, for all test suites. As all tests were performed using the same version of the SUT, the initial failure density (sum of detected and undetected faults) could be compared. Random test suites with different coverage and of different lengths were executed and the results were compared. The expected outcome (from this random process) is a normally distributed estimation of the initial fault density, with the mean close to the true value. Moreover, the proportion of the detected faults are expected to be a function of test coverage and length of the test suites.

However, in terms of coverage, the samples are not as evenly spread as desired. The coverage turned out to be hard to control with high precision.

The gathering of samples (test suites) takes a long of time (months), more than 3 million tests were performed. This has limited the number of test suites to 85. Ideally, more versions of the software should have been tested. Further studies of other types of software and of different criticality would be of great interest. The aspect of criticality of the software, is believed to be of importance to the eveness of the fault density throughout the system. The SUT in this case is classified as being of low/medium criticality. Which means that there is less formality and rigor in the development process.

The variance in the estimates is expected to be less for software of higher criticality, e.g., DAL A and ASIL D. This will be investigated in further studies. A conclusion that can be drawn is that the results from Equation 2 are coherent and

has a consistent distribution. When these results are used to calculate the entropy the results fit well to the theoretical data.

VIII. Conclusion

This paper analyzes quantification and interpretation of test results for safety-critical software.

A uniform sampling of the input space, i.e., all values of each parameter is given the same probability of occurrence, allows for prediction of the faults that remain undetected in the system after a (long) test suite. This paper shows that predictions can also be made if only parts of the parameter ranges are sampled, i.e., a defined subspace is sampled.

The modified Equation 2, provides a good prediction of undetected faults for test suites with a limited test coverage. Results indicate that, for this purpose, code coverage data can be used as an approximation to limitations of input space sampling. Data from the industry case study show good correlation for both calculation of failure density and entropy.

For more complex input spaces, e.g., systems that consider signal history, it can be difficult to determine how much of the input space that is sampled during the generation of input. The industry data shows that code coverage data can be used as an approximation of the input space coverage, for the purpose of calculating failure density and entropy.

Today, safety-critical software is developed according to certified processes and by following strict guidelines. It is not required to mathematically assess the failure rate of the software. However, the concept described here, of structured testing where the results are treated as a statistical sample, allows for a quantification of the status of safety-critical software. Both the entropy- and the failure density concept are correlated to failure rate of the software and can be implemented and used by the industry, for pre-deployment risk assessment. The concept can summarize the compiled test results from the implemented test process or alternatively be used together with (additional) random testing.

Acknowledgement

This work was partly supported by The Swedish Knowledge Foundation (KKS) through the research profile Dependable Platforms for Autonomous Platforms and Control (DPAC), and the Industrial Graduate School for Reliable Embedded Sensor Systems (ITS ESS-H).

References

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

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

[3] J. A. McDermid and T. P. Kelly, "Software in safety critical systems: Achievement and prediction," *Nuclear Future*, vol. 2, no. 3, pp. 140–146, 2006.

[4] B. Littlewood and L. Strigini, "Validation of ultra-high dependability - 20 years on," *Safety Systems - The Safety-Critical Systems Club Newsletter*, vol. 20, May 2011.

[5] A. Bertolino and L. Strigini, "Assessing the risk due to software faults: Estimates of failure rate versus evidence of perfection," *Softw. Test., Verif. Reliab.*, vol. 8, no. 3, pp. 155–166, 1998.

[6] J. Sundell, R. Torkar, K. Lundqvist, and H. Forsberg, "Prediction of undetected faults in safety-critical software," in *2nd IEEE Workshop on NEXt level of Test Automation*, April 2019.

[7] D. L. Parnas, A. J. van Schouwen, and S. P. Kwan, "Evaluation of safety-critical software," *Commun. ACM*, vol. 33, pp. 636–648, June 1990.

[8] 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, (New York, NY, USA), pp. 89–98, ACM, 1993.

[9] B. Littlewood and L. Strigini, "Validation of ultrahigh dependability for software-based systems," *Commun. ACM*, vol. 36, pp. 69–80, Nov. 1993.

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

[11] J. Voas and K. W. Miller, "Software testability: The new verification," *IEEE Software*, vol. 12, pp. 17–28, 1995.

[12] A. Bertolino and L. Strigini, "Using testability measures for dependability assessment," in *Proceedings of the 17th International Conference on Software Engineering*, ICSE '95, (New York, NY, USA), p. 61–70, Association for Computing Machinery, 1995.

[13] A. Bertolino and L. Strigini, "On the use of testability measures for dependability assessment," *Software Engineering, IEEE Transactions on*, vol. 22, pp. 97 – 108, 03 1996.

[14] A. Bertolino and L. Strigini, "Acceptance criteria for critical software based on testability estimates and test results," in *Safe Comp 96* (E. Schoitsch, ed.), (London), pp. 83–94, Springer London, 1997.

[15] J. M. Voas, C. C. Michael, and K. W. Miller, "Confidently assessing a zero probability of software failure," in *SAFECOMP '93* (J. Górski, ed.), (London), pp. 197–206, Springer London, 1993.

[16] A. Arcuri, M. Z. Iqbal, and L. Briand, "Random testing: Theoretical results and practical implications," *IEEE Transactions on Software Engineering*, vol. 38, pp. 258–277, March 2012.

[17] 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*, pp. 308–317, Oct 1996.

[18] P. G. Bishop and R. 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, (USA), p. 295, IEEE Computer Society, 2002.

[19] D. Clark, R. Feldt, S. Poulding, and S. Yoo, "Information transformation: An underpinning theory for software engineering," in *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, (Piscataway, NJ, USA), pp. 599–602, IEEE Press, 2015.

[20] S. Mankefors, R. Torkar, and A. Boklund, "New quality estimations in random testing," in *Proceedings of the 14th International Symposium on Software Reliability Engineering*, ISSRE '03, (Washington, DC, USA), pp. 468–, IEEE Computer Society, 2003.

[21] A. Jedlitschka, M. Ciolkowski, and D. Pfahl, "Reporting experiments in software engineering," in *In Guide to Advanced Empirical Software Engineering*, pp. 201–228, Springer, 2008.

[22] B. T. Technology, "Bullseye product description," August 2019. Accessed: 2019-08-23.

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

[24] P. Bishop and R. Bloomfield, "Using a log-normal failure rate distribution for worst case bound reliability prediction," pp. 237– 245, 12 2003.

[25] R. E. Mullen, "The lognormal distribution of software failure rates: origin and evidence," in *Proceedings Ninth International Symposium on Software Reliability Engineering (Cat. No.98TB100257)*, pp. 124–133, 1998.

[26] T. G. Grbac, P. Runeson, and D. Huljenic, "A second replicated quantitative analysis of fault distributions in complex software systems," *IEEE Trans. Software Eng.*, vol. 39, no. 4, pp. 462–476, 2013.

[27] N. E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system," *IEEE Trans. Softw. Eng.*, vol. 26, pp. 797–814, Aug. 2000.