On some statistical aspects of software testing and reliability

Frank P.A. Coolen

Abstract This article discusses the author's views on possible contributions statistics can make to software testing and reliability. Several difficulties are highlighted and several research challenges are discussed. Overall, the message is that statistical methods cannot provide proper support for software testing or provide useful inferences on software reliability if the statistical methods are considered to be an 'add-on'; careful treatment of the uncertainties and the adequate use of statistical methods have to be right at the center of the software development and test processes to ensure better tested and more reliable software. In line with this requirement, the development of suitable statistical methods requires collaboration of software developers and testers with statisticians on real-world problems.

1 Introduction

Ten years ago, a paper entitled 'Bayesian graphical models for software testing', written by my colleagues David Wooff, Michael Goldstein and me, was published in IEEE Transactions on Software Engineering [16]. This presented the pinnacle of a substantial multi-year research project in collaboration with an industrial partner, in which we explored the possibilities to use statistical methods to support software testers. As testing software is effectively all about uncertainty and information, it seemed obvious that statistics, which may be regarded as the art of dealing with uncertainty and information, could help testers with their very complex tasks. This project was successful, particularly in setting a direction for future collaboration between statisticians and software testers; an overview of the project was presented in the

Dept. of Mathematical Sciences, Durham University, Durham, United Kingdom
`frank.coolen@durham.ac.uk`

paper ‘Using Bayesian statistics to support testing of software systems’ (by the same authors), published in the Journal of Risk and Reliability [8].

After initial difficulties due to very different jargons and cultures between the academic statisticians and the real-world software testers, very useful meetings followed during which the software testers explained more and more which aspects involving uncertainty were really difficult, what their current practice was with regard to all aspects of software testing, and which specific aims and restrictions there were for the software testing. It should be emphasized that the ‘current practice’ was considered to be good, certainly in line with the state-of-the-art at the time of the project (late 1990’s), and indeed that there were quite many and different aims, particularly when managers with different responsibilities in the company got involved and expressed the hope that the methodology we were developing in collaboration with the software testers would also be useful to assist them in their specific duties related to the software testing process. For example, one such a duty involved setting the budget for a specific software testing project, well in advance of the software actually becoming available. I refer the interested reader to the above cited papers for more details on the specific project, and particularly on the Bayesian graphical modelling approach we developed. A substantial monograph that addresses many more aspects of that project, including for example methodology to predict the time required for software testing including considerations of re-testing after failures, and indeed how to do such re-testing efficiently, is in preparation [17].

In this article, I reflect a bit further on aspects from the mentioned long-term project, and on research questions that arose from it. Beyond this, I reflect on several aspects of the interaction between statistics and software testing and reliability, raising some important challenges for research and application that, to the best of my knowledge, are still open topics. This article does not answer many questions, but I hope that it provides some possible directions towards answers, or at least some issues to reflect upon and which might steer future developments and collaborations in the field.

2 Statistics, software testing and reliability

The question how statistics can help software testing and be used to assess software reliability appears, at first, a simple question. However, it is not. To start, we have to consider what statistics is, and indeed what software testing and software reliability are. These are all generic terms that encompass many different activities, problems, methods, theories and so on. Generally speaking, however, statistics can be regarded as the theory of quantification of uncertainty, which includes the effect of information on uncertainty. As such, it is natural to use statistics to support all software testing and software reliability problems where dealing correctly with uncertainty and information is a major issue. It should be emphasized that not all problems in testing and software require, or can benefit from, the use of statistical methods. For example, if one is absolutely certain that checking code will reveal all possible problems, then one has no need to quantify uncertainty and therefore statistics has nothing to offer. Of course, the moment the 'absolute certainty' disappears then one should consider the use of statistical methods.

In statistics, there are two predominant foundational frameworks, which some might see as competing theories yet they both have their own important place in uncertainty quantification in general, hence also with regard to software testing and reliability. The frequentist framework of statistics generally provides methods that are calibrated in case of repeated use, under suitable assumptions. Results are traditionally in terms of e.g. confidence or significance levels, which are difficult to interpret for specific applications and indicate an approximate proportion of times that the specific method, if applied to many different problems, will give the correct answer. While this has been criticized by many, it is not a bad principle: if one has a tool available which is known to deliver a good outcome for a specific kind of task approximately 95 out of 100 times it is being applied, many will consider it a suitable tool. One problem with many of the classical frequentist approaches is that uncertainties often involve unobservable quantities such as model parameters, which add a further level of difficulty to understanding of the results in many cases.

The Bayesian framework of statistics has very different foundational starting points, with emphasis on subjectivity of probability to quantify uncertainty jointly for all random quantities in a specific application, where inferences follow from probabilistic conditioning on the information that becomes available. There is no frequency interpretation as a starting point, but in practice results from Bayesian statistics are often either fully or approximately in agreement with their frequentist counterparts. This is not surprising when there is a substantial amount of data and the wish not to let modelling assumptions or additional information in the form of expert opinions influence the inferences. However, in situations with few data the methods provide quite differing opportunities, where the explicit inclusion of expert opinions in the Bayesian framework can be a substantial advantage. It should be remarked that both the frequentist and Bayesian methods can be generalized by the use of sets of probabilities instead of a single one, which provides more robust inferences and models more adequately any lack of information. In particular some generalized methods for uncertainty quantification provide attractive opportunities to take unobserved or even unknown possible outcomes or risks into account [1, 2, 4, 11]. Such imprecise probabilistic methods are gaining popularity but have thus far not yet been implemented for software testing [9]. For general reliability problems, including a little attention to basic software reliability, such methods have been developed [10].

Software testing, and related aspects of software reliability, vary widely in different applications. The number of inputs that can be tested can vary from very few to effectively an unlimited number; some functionality might not be testable; it may be anything from trivial to impossible to know if an output is correct; the tests may take place on the actual software system or on a special test system which may resemble but not be identical to the actual system; the software may be fully open for inspection, it may be entirely black-box or anything in between, et cetera. Furthermore, the effects of errors in the output may be anything from neglectable to catastrophic, and may even be genuinely unknown. Statistical methods can be used to support decision making in all such scenarios, but it requires a high level of statistical expertise to ensure correct choice and application of methods. Importantly, it must be emphasized that statistics is not a miracle cure, it can only deal with the information that is available and its use should be aimed at supporting software testers.

The power of statistical methods lies mostly in careful learning from information and guidance with regard to further testing, which involve problems that even for basic test scenarios are soon too complex for the human mind to solve without the aid of sound statistical methods. Dealing correctly with uncertainties in applications is, in most cases, difficult and time consuming. The benefits can be substantial but of course will require justification with regard to the effort. Such considerations may be used to guide the level of detail of statistical modelling. This necessarily must be done related to a specific statistical approach; as an example of such guidance I refer to [7] where the considerations for deciding an appropriate level of detail in Bayesian graphical models for software testing are discussed.

In the following section, I will discuss a number of topics on the interface of statistics and software testing and reliability, which I believe require thought and attention, and indeed further research. It is crucial that such further research is linked to real-world applications with genuine collaboration between statisticians and software testers and engineers. These topics are discussed briefly without attempts to provide a full picture of the problems and state-of-the-art. It will be followed by a brief personal view on the way ahead in Section 4.

3 Some topics that require attention

The first thing to consider in software testing is what actually can be tested. This is particularly important if testing leads to quantification of reliability of the software system in practical use. Usually, testing is limited in scope and executed during a relatively short period of time, possibly even without full linkage of a specific system with other real-world systems, databases et cetera. Statistical methods using only information from such limited tests cannot be expected to extrapolate to more practical issues. Software systems may also fail due to failure modes occurring at the interface between software and hardware, e.g. electricity provision might be interrupted during software processes, or due to background activities (e.g. memory space may slowly fill up); such aspects often cannot be discovered during practical testing, hence deducing a level of reliability from test results requires care.

A very basic question which, to our surprise, took quite some time to resolve in the aforementioned application of Bayesian graphical models to support software testing, is what to model? In some applications, particularly where very many tests can be done, one might simply be able to consider the input domain and model for each element of it (there may be 'infinitely' many, for example if inputs are real-valued quantities) whether or not the software provides the correct output. The software testing we were involved with, however, considered integration testing of new or upgraded functionality and code, particularly with regard to communication between databases. Inputs were typically related to some processes, and customers could possibly provide all kinds of inputs and in different orders, so it was difficult to define a theoretical input space. Testers would create scenarios to go through in their testing, and of course such test scenarios reflected their expertise in detail. We modelled the testers' knowledge and beliefs about the software system and its functioning, in particular how they distinguished between related but different inputs and where they believed possible errors in output would originate from. The software systems were complex and (almost entirely) black-box, with parts developed over quite many years without detailed knowledge about the development and core software remaining in the company. The statistical methodology was then used explicitly to assist the testers in their extremely complex tasks with regard to selecting efficient tests, ordering these, inferring what to do if a test outcome was wrong, and indicating when testing could be stopped[1]. As the statistical approach clearly modelled the testers' expertise and actions and enhanced their expertise, without taking over (and it was clear to the testers that the approach would not be taking over their roles in the future, due to the individual aspects of specific software systems and testing tasks), they understood the supporting role of the statistical methodology, helping them to do their job better and more efficiently. This is crucial for any collaboration, of course, but perhaps particularly so when experts are sceptical about novel methods that claim to be able to support them but which they may find difficult to understand. Actually, we found that the software testers quite quickly got to grips with the Bayesian graphical models that we built, and with the possibilities of support these gave them. They also understood well that the method could only provide useful support if they included useful and honest information on which to base the models. A lot more research is required on this topic, particularly considering a wide

[1] In early papers in the software testing literature the view was sometimes expressed that testing was of no use if it did not reveal failures. This is quite a remarkable view, as no failures is of course the ultimate aim. Probably the more important question is when to stop testing in such cases, this links to the topic of high reliability demonstration where, of course, statistical methods can also provide guidance [5].

variety of software testing and reliability scenarios, and it is hard to say which aspects can and should be modelled without considering a specific scenario.

An important consideration in practical testing is what the consequences will be of a software error, and indeed if such errors can be recognized. In the project we were involved with it was not a problem to discover software errors. Generally, if it is not known with certainty whether or not software output is correct (think e.g. at complicated computational software, particularly cases were computation for one input can take a very long time), Bayesian statistical methods can provide some modelling opportunities, mostly by comparing results with prior expectations for them and trying to identify if results differ from expectations by a substantial amount. I have not yet seen detailed applied research about this, it will be fascinating and important.

Consequences are often, although not necessarily explicitly, grouped into e.g. cosmetic, minor, major or catastrophic. Typically, under severe time and costs constraints, tests are not aimed at discovering failures with cosmetic and perhaps even minor consequences, but any such errors are noted and dealt with at a possible future opportunity. Test suites designed by testers normally reflect the severity of consequences of failures, particularly so if such failures occurred in the (recent) past. Stastical methods can be set up in order to deal with different consequences of failures, and hence the importance of discovering such failures. The concept of 'utility' is a natural part of the Bayesian framework and is precisely used to take such consequences into account. However, in a company with different levels of management, with several different people responsible for different aspects of the output of the software system that is being tested, such utilities may not be discussed explicitly and they may differ based on the personal views and responsibilities of specific people. What to some people might appear just a failure with minor consequences, could by others be regarded as a major problem or worse. Such aspects must be discussed as it is crucial that there is a unified and clear goal for the software testing. One may be tempted to define 'delivering fault-free software' as such a goal, but this is mostly unrealistic for substantial software systems and would be extremely difficult to verify from test results, unless time and budget for testing are extremely generous. When using statistical methods to support software testing, the utilities related to the outcomes, and indeed to finding failures in order to prevent future problems with specific outputs, are important and must be determined at a relatively early stage in the process as they will guide the focus and level of detail of the testing and therefore of the statistical modelling process.

An intriguing situation occurs if one may not know (expected) possible consequences of failures, or may not know what kind of failures can occur. This is very natural, in particular when testing new black-box systems where discovering the functionality may be part of the testing aims. Statistical methods that can support such testing have not really been developed, and hence this provides very important and interesting research challenges. Imprecise probabilistic methods, including nonparametric predictive inference [4], have some specific features which make them promising for inference on the occurrence of unknown events. The theory of decision making with utilities has also been nearly entirely restricted to known outcomes and known utilities. Recently, a generalization to adaptive utilities has been presented, both within the Bayesian and nonparametric frequentist statistical frameworks [13, 14, 15]. In this work, utility is uncertain and one can learn about it, which of course is quite often a major reason for experimenting and testing (e.g. to learn about unknown side-effects of medication; there is a suitable parallel to software). We have ideas to link this work to software testing, but it would be from theoretical perspective at first; we hope to link it to real-world test situations in the future.

As mentioned, there may be several managers in an organisation who each have different responsibilities and hence may have different needs for statistical methods to support their activities related to software testing and reliability. For example, someone managing the actual test team will have quite different specific needs for support compared to someone managing the overall development and test budgets, and indeed having to make cases to set such budgets, with the latter typically needing to make decisions quite far in advance to the actual testing. Statistical methods, in particular the Bayesian framework, can support all such different decision processes, but again this may require problem specific modelling, will not be easy and will not be a miracle cure. As part of our industrial collaboration[2] we developed an approach to provide, approximately, an expected value and corresponding variance for the length of a future testing process, supported by the Bayesian graphical modelling approach. This approximation took into account re-testing after corrections of software failures discovered during testing. Of course, this requires substantial input as expected times must be provided for all possible events together with information on probabilities, not only for failures to occur but also with regard to success of correcting actions. Managers should not expect statistical methods to be able to provide answers to such difficult questions without such inputs, it underlines that careful modelling is difficult and time consuming, as is generally the case if one wants to deal well with uncertainty.

In addition to such quite general aspects there are a number of issues that, quite remarkably, still do not seem to have been solved satisfactorily, or at least about which there is substantial confusion amongst software testers and even some researchers who present methods with a suggestion of statistical expertise. For example, if one can do ample testing and has detailed knowledge of the input profile for the software system in daily practical use, it tends to be advocated that testing should be according to that input profile. However, this will rarely be optimal use of test time and budget (unless these are effectively limitless). Intuition on a most simple example probably suffices to explain this point: suppose that a software system will only have to

[2] This will be reported in the monograph [17] that is being prepared

provide outputs for two groups of inputs, say groups A and B. Suppose that the input profile from practical use of the system is such that 99% of inputs are from group A and only 1% from group B. Suppose further that 100 inputs can be tested. In this case, testing in line with the input profile would lead to 99 inputs from group A and one input from group B to be tested. Intuitively it will be clear that the 99 tests of inputs from group A probably leave little remaining uncertainty about the success rate of the software on dealing correctly with inputs from this group. But the single test from group B is not so informative for other inputs from this group. Hence, if one wishes to have high quality software which rarely makes mistakes, then one would probably wish to do some more tests for group B and reduce the number of tests from group A accordingly. It is not difficult to prove this mathematically, but the crucial part is to recognize that the input profile is to be used related to the optimality criterion, that is proportions of applications in the future, after the testing, with related utilities for avoiding failures in the outputs. Then the optimisation variables should be the proportions of inputs from each group within the test, and this will rarely lead to testing in the same proportions as the input profile [3].

In this reasoning with regard to profiles there is an aspect that is at the heart of software testing and the statistical support for it, namely the required judgements on exchangeability of the different inputs [**?**, 12]. A key judgement that software testers have to make, and that must be reflected in any statistical model and method to support them, is how similar different inputs are, for a specific software system, with regard to the expected quality of the corresponding output, and with regard to what knowledge about the corresponding output quality reveals about such output quality for other inputs. One extreme situation would be that the software's output either will be incorrect for all inputs or will be correct for all inputs. Under such judgement, clearly a single test suffices if one can correctly classify the output. The other extreme situation would be that every tested input only reveals whether or not the software provides the correct output for that specific input value, while not revealing anything about other inputs. In the first case the software's performance is identical for all inputs, in the second case it is independent for all inputs. In practice the truth, and with it the assumptions testers are confident to make, tends to be somewhere between these two extremes. Statistically, the concept of exchangeability (and partial exchangeability [12]) provides the opportunity to model such judgements, and these can be taken into account in a variety of ways in statistical methods. However, it is not easy to do so, this is often overlooked [3]. In particular, one might naturally judge that the input space can be divided into a partition, with all inputs in one element of the partition being more similar than inputs in different elements, but with neither being identical or independent. Bayesian graphical models provide a modelling framework for this, but are not as flexible or easy to deal with in case of learning from many test results as some possible alternatives [6]. Further research is required on this crucial aspect which sits at the heart of uncertainty in software testing.

Practical statistical support for software testing highlights one major research challenge for statistics which, perhaps remarkably, appears not yet to have attracted much attention. Any decisions on design of test suite (and beyond software testing, general design of experiments) depend on modelling assumptions, usually based on expert judgements, experience and convenience of the corresponding statistical methods. Ideally, one would want to use the test suite also to confirm the underlying model assumptions, in particular where tests are performed sequentially this could lead to changes if it was discovered that the assumptions were not fully adequate. In such cases, one would still hope that the earlier test results would be of use while one could adapt further tests in the light of the new information, and so on. This idea to use test outcomes in case the underlying assumptions were not fully adequate is an issue of robustness of the test suite with regard to aspects of the model specification. The idea to take this into account is particularly relevant for sequencing of tests, where at the early stages of testing one may want to include tests which can indicate whether or not the modelling assumptions are adequate. It has, to the best of my knowledge, not been considered in the literature, on software testing and even on general design of experiments. A small step to this idea is the concept of adaptive utility [13, 14] in Bayesian methods, where in sequential decision processes it is shown that it can be optimal to first look for observations that enable one to learn the utilities better, to the possible benefit of later decisions. Due to the specific features of software testing, it would be exciting if this aspect of designing suitable test suites were investigated in direct connection to a real-world application.

4 The way ahead

I should make an important comment about the (mostly academic) research literature on software testing and reliability: In practice there is often scepticism about the methods presented, and I believe rightly so. Most of these methods have been developed by mathematicians and statisticians based on assumptions that are inadequate in many challenging software testing problems. For example, the many models based on the idea of fault counting and removal or on assumed reliability growth appear not to have a substantial impact on practical software testing. Our industrial collaborators, who were very experienced software testers and engineers, did not even think in terms of numbers of faults in the software, as it was largely black-box and defining a fault would be non-trivial.

I believe that the only way ahead is through genuine collaboration between software testers and statisticians throughout the process of testing, including all stages of test preparation. This is difficult and might be expensive, so might only be considered feasible for substantial applications, particularly with safety or security risks in case software failures occur. However, sound statistical support for the software testing process is likely to lead to both more efficient testing and more reliable software, as such it will probably lead to cost reduction, both for the actual test process and with regard to the consequences of software failures. It would be particularly beneficial to have long-term collaborations between the same teams of testers and statisticians, both for working on upgrades of some software systems and for working on a variety of systems. Testing upgrades normally benefits greatly from experiences on testing of earlier versions, while working on a variety of systems will also provide great challenges for the testers and statisticians, which is likely to provide useful further insights towards more generic approaches for statistically supported software testing.

I strongly hope that during the next decade(s) a variety of such long-term collaborations will be reported in the literature and will lead to important further methods and insights. Possibly some generic aspects might be automated, to facilitate easier implementation of sound statistical support for testing of software with some specific generic features. I am sceptical however about the possibility to fully automate such statistical support. I am sceptical about the possibility to fully automate software testing, and as dealing adequately with the uncertainties involved adds substantial complexity to the problem, full automation is highly unlikely. It is beyond doubt, however, that thorough long-term collaborations between software testers and statisticians can lead to very substantially improved software testing, hence leading to more reliable software which will justify the effort. Clearly, this is a great field to work in due to the many challenges and opportunities, both for research and practical applications which have to be developed together.

Acknowledgements

I am grateful to the members of the organising committee of the Seventh International Conference on Dependability and Complex Systems (DepCoS-RELCOMEX 2012), in particular Professor Wojciech Zamojski, for their kind invitation to share my experiences and views on the interaction between statistics and software testing and reliability with the conference participants, both through a presentation at the conference and this article. The contents of this contribution are fully from the author's perspective, but are based on several longstanding and ongoing research collaborations, as reflected by the bibliography of this article. I sincerely thank all collaborators for their substantial inputs in jointly authored research publications, and beyond that for the many discussions which have played a major part in shaping my research activities and views on interesting research challenges.

References

1. Coolen FPA (2006) On probabilistic safety assessment in case of zero failures. Journal of Risk and Reliability 220:105–114.
2. Coolen FPA (2007) Nonparametric prediction of unobserved failure modes. Journal of Risk and Reliability 221:207–216.
3. Coolen FPA (2008) Discussion on: 'A discussion of statistical testing on a safety-related application' by Kuball and May. Journal of Risk and Reliability 222:265–267.
4. Coolen FPA (2011) Nonparametric predictive inference. In: International Encyclopedia of Statistical Science; Lovric (Ed.). Springer, Berlin, 968–970.
5. Coolen FPA, Coolen-Schrijner P (2006) On zero-failure testing for Bayesian high reliability demonstration. Journal of Risk and Reliability 220:35–44.
6. Coolen FPA, Goldstein M, Munro M (2001) Generalized partition testing via Bayes linear methods. Information and Software Technology 43:783–793.
7. Coolen FPA, Goldstein M, Wooff DA (2003) Project viability assessment for support of software testing via Bayesian graphical modelling. In: Safety and Reliability; Bedford, van Gelder (Eds.). Swets & Zeitlinger, Lisse, 417–422.
8. Coolen FPA, Goldstein M, Wooff DA (2007) Using Bayesian statistics to support testing of software systems. Journal of Risk and Reliability 221:85–93.
9. Coolen FPA, Troffaes MCM, Augustin T (2011) Imprecise probability. In: International Encyclopedia of Statistical Science; Lovric (Ed.). Springer, Berlin, 645–648.
10. Coolen FPA, Utkin LV (2011) Imprecise reliability. In: International Encyclopedia of Statistical Science; Lovric (Ed.). Springer, Berlin, 649–650.
11. Coolen-Maturi T, Coolen FPA (2011) Unobserved, re-defined, unknown or removed failure modes in competing risks. Journal of Risk and Reliability 225:461–474.
12. De Finetti B (1974) Theory of probability (2 volumes). Wiley, London.
13. Houlding B, Coolen FPA (2007) Sequential adaptive utility decision making for system failure correction. Journal of Risk and Reliability 221:285-295.
14. Houlding B, Coolen FPA (2011) Adaptive utility and trial aversion. Journal of Statistical Planning and Inference 141:734-747.
15. Houlding B, Coolen FPA (2012) Nonparametric predictive utility inference. European Journal of Operational Research, to appear.
16. Wooff DA, Goldstein M, Coolen FPA (2002) Bayesian graphical models for software testing. IEEE Transactions on Software Engineering 28:510–525.
17. Wooff DA, Goldstein M, Coolen FPA. Bayesian graphical models for high-complexity software and system testing. Springer, in preparation.