The Future of Testing for Safety Critical Systems



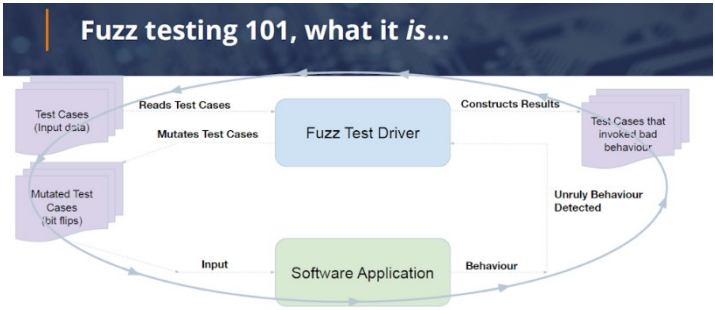
The "Future of Testing for Safety-Critical Systems" seminar was held on the 1st December 2022 at Hard Rock Hotel, London and virtually online. The seminar examined how safety-critical systems are tested. It covered tools, techniques and limitations of testing methodologies and also explored some historical failures of testing and the lessons learnt.

Mike Parsons introduced the seminar and the seminar speakers and said testing was a topic close to his own heart but the topic was vast and applicable at many different levels and there are many techniques and ways of testing. Automation is also a hot topic and there are particular challenges with Artificial Intelligence/Machine Learning (AI/ML).

Fuzzing for High Integrity Software

Paul Butcher from AdaCore, discussed work he was doing in Fuzz Testing under the High Class research initiative – a UK joint venture between industry and the government focussing on cybersecurity in civil avionics. Paul introduced the concept of Fuzz Testing as the injection of random data into a process, which has its origins in the 50's.

Now very popular in Automotive, less so in Defence and Aerospace, but this is changing now due to legislative changes that are helping drive the concept through advisory circulars and guidelines coping with unauthorised electronic access. Advances in hardware allow higher volumes and complexity of data to be injected into the system.



Paul introduced the open source American Fuzzy Lop (AFL) (now AFL++) tool that has a mutational approach, and AdaCore is a maintainer of this source.

Paul described what Fuzzy testing is and said it is more focussed on system behaviour (is in operating in a known state, not crashing etc) rather than checking detailed outputs. The Fuzzer takes an initial input and mutates these at a binary level (without knowledge of structure). One question is how long fuzz testing should be run for, that is, the stopping criteria, and Paul referred to the Security Refutation Objectives in "Airworthiness Security Methods and Considerations" (EUROCAE ED-203A/RTCA DO-356A).

Primary objectives are to automate test case generation, execution and anomaly detection (segmentation faults, hung processes and other language runtime constraint checks: eg. overflow violations). Other more advanced aspects can be techniques such as differential-fuzzing where outputs can be checked by using a reference model and algorithmically generating expected outputs.

Mutation Strategies are generally better than pure random selection and involve making small changes to an initial set and there are different approaches: deterministic, random, splice, dictionary (help to ensure mutations don't get thrown away) and custom.

Fuzzing testing can borrow from genetic algorithms or "smart-grey-box" fuzzing. The natural selection element is 'did my test make progress', for example, by improving the test coverage.

Paul discussed some state-of-the-art approaches:

- Input-to-State Correspondence (also known as "Red Queen" or "CmpLog") helps solve problems with getting into simplistic "tight branches" where entry into the branch is hard, purely by selecting random parameters
- Symbolic Execution using tools like KLEE, is a more heavy-weight approach for the same problem but where there is no clear input to state correlation. Symbolic execution tries to execute multiple code paths simultaneously by working out what the inputs should be to arrive at a leaf node in the execution tree. This does suffer from path explosion and memory issues.
- Concolic using tools like QSym/SymCC is a combination of concrete (traditional unit testing) and Symbolic (especially when coupled with fuzzing)
- Differential Fuzzing is where outputs resulting from fuzzed inputs are compared with a reference model with outputs being compared.

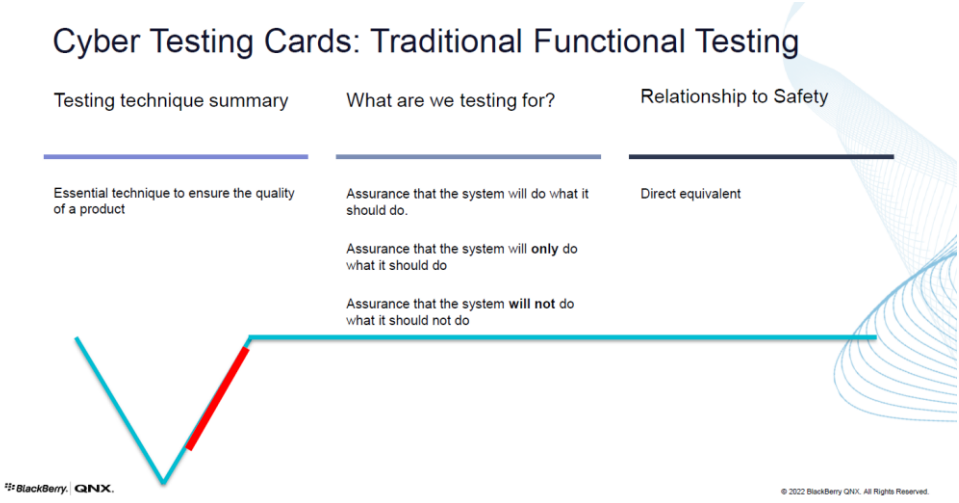
Cyber Security Testing Techniques

Bob Oates from Blackberry QNX – a security company with a focus on automotive and medical device operating systems but with a pedigree in both safety and security.

Bob started with the key security test challenges:

- Quality: (you can't do safety/security without an underpinning of quality. Also test-driven development; breaking the architecture up so it is easier to test
- Technology trends: hyperconnectivity with lots of systems of systems (eg. software-defined vehicle) lots of integrated subsystems. Challenge of security and test coverage especially in autonomous vehicles.
- Threat Trends: increase in attacks of critical national infrastructure and in general state sponsored cyber warfare
- Security Role: safety/security no longer can be siloed
- Resource Constraints: working in a constrained environment and so test worthiness is important

Bob then discussed a number of Cyber Testing Cards:



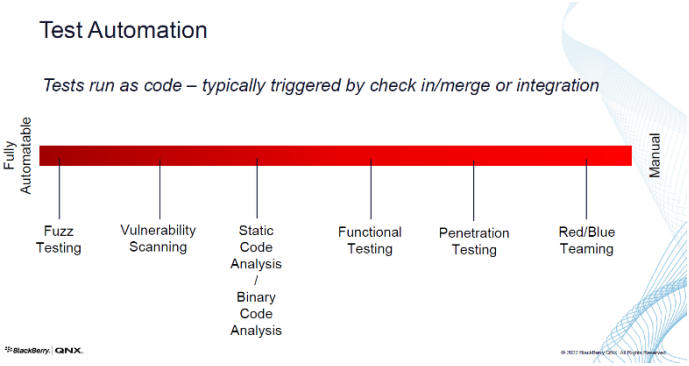
These cards covered:

- **Traditional Functional Testing**: testing against well-define requirements that the system will behave only as specified (classic right-hand side of the "V Model")
- **Fuzz Testing:** scalable approach providing mutated inputs and recording behaviour – highly automatable to test for unexpected behaviour
- **Static Code Analysis:** classic cyber security testing technique, checking code against pre-defined rules such as those in MISRA C and MISRA C++
- **Binary Code Analysis**: decompile binary code using tools like Jarvis and look for 3rd party library usage, static strings and software bill of materials to do license compliance and check against product vulnerability libraries

- **Active Vulnerability Scanning**: scan systems for open ports and library use and mis-configuration. Dangerous to run on live operational system as tests can be harmful!
- **Penetration Testing**: Simulating an attacker with a predefined anchor into the system (such as giving the attacker a user's logon credentials). Need well-defined rules of engagement to ensure the testing is not interacting with safety-critical systems
- **Red v Blue Team Testing**: a more modern technique with a blue team also being part of the test acting in an operational capability to detect and respond to the attacker (red team). Bob emphasised that the blue team can be just as dangerous as the red team if they take an inappropriate action.

Bob then discussed Test Automation and said that it needs a well-defined environment (eg. through use of tools like Docker to automatically provision environments) and well-defined pass/fail criteria.

The level of automation varies depending on the technique and Bob proposed the scale given in the figure opposite.



Bob also said the security of the IT environment, such as the development environment tools, is equally important.

Bob concluded by saying:

- Security is going to be a huge part of the future of safety testing
- We need to understand when and how to perform security tests
- Automation is going to be key to achieving "Test worthiness"

Testing in the real world: hard lessons learnt

Ian McDonald of Island Systems Ltd and John Dixon of Platinum QA Systems started their talk with a description of the costs of failure of software projects that have led to deaths, huge economic impact, costly repairs and recalls and brand damage. Despite this, Ian said it is often a battle to ensure people take a responsible and ethical approach to ensuring systems are sufficiently well-tested.

Ian provided some real-world examples of their experiences, for example, a case where software development was subcontracted without the contract defining the success of code coverage. Requirements were also long with many embedded sub-requirements so difficult to assess test success. This resulted in defect fixes leading to even more defects and the delivery falling behind schedule.

It is key to understand why a customer wants something, their expectations, the timescales, the level of required quality of product and also the stakeholder's perception of risk.

In some cases, an element of evangelism is required to promote the benefits of properly documenting and accurate capture of metrics such as root cause analysis. Blame culture can also be detrimental to a project development as is incentivising the absence of defects as this can lead to lack of reporting and gaming of the test systems.

Hardware testing also needs to be considered – there is a general focus on software testing but issues with the hardware can be just as serious.

Lack of formalised QA and test approaches lead to poor estimation and hence delayed projects that are over budget and bug ridden. John said that testing does not start at the end of a development but at the beginning by ensuring that the requirements make sense, are atomic and are testable.

John and Ian also said to be wary of standard metrics for estimating the amount of testing on a project (eg. 40% of total effort) as large projects require much more effort due to the larger number of interactions.

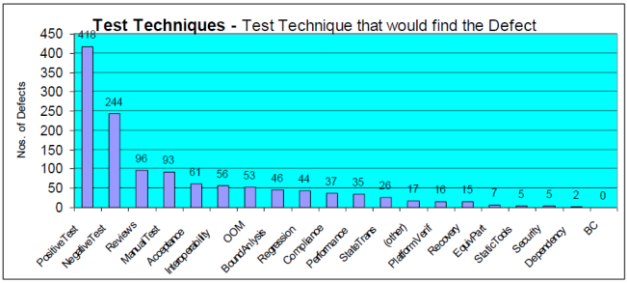
There are techniques to allow you to develop the least number of tests for the maximum impact called boundary value analysis focussing on tests of values around the minimum and maximum values that a variable can hold.

Test strategies are often too reliant on things happening perfectly. A good test manager will be thinking about alternative paths if the unexpected occurs.

Care should also be taken when making assumptions about relying on testing undertaken when reusing software in a different context and gave an example of the Ariane 5 launch failure.

Automation is not a magic bullet; it can help to speed repeatability of testing but there is a cost for updating automated tests and this can take longer than writing the code. Automation doesn't replace good test design. Poor automation may fail to provide sufficient coverage and maintenance costs can mount.

Code standards and implementing code review standards is seen as important. Reviewing code is cheap but it is important that reviews are frequent and have the right people in the review.



Ian concluded by asking how problems are fixed. Two key techniques were discussed: Root Cause Analysis (RCA) and Test Escape Analysis (TEA). Ian said that RCA feedback is often provided late and is therefore not that effective. TEA looks at why defects escaped early detection. This involves a periodic review between the test and development leads to classify defects with the aim to feed this information back into test planning. The figure shows an example chart of defect counts using this method.

Bad Behaviour: Automating system-level safety testing

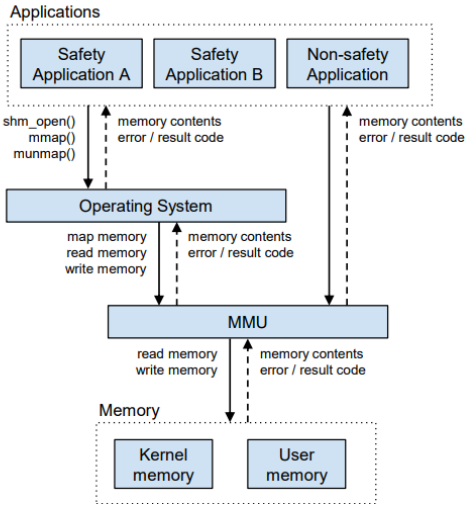
Paul Albertella from Codethink, opened his talk with a discussion of his background in using open source in automating development and testing of systems. Paul said that software testing plays many essential roles in safety. Different types of test have different purposes, context and scope, for example, not just verifying the correct implementation but also other aspects such as detecting issues with integrating software, validating safety mechanisms, testing systems under stress and under adverse environmental conditions.

Paul said that system-level testing is important as safety, being an emergent system property, can only be understood by considering how all the system components work in concert together in a particular context – especially timing aspects.

There are a number of challenges with testing for safety:

- identifying "bad behaviour" rather than just the "happy paths"
- understanding the requirements tests are intended to verify
- ensuring test completeness – coverage does not mean that all conditions have been tested
- making sure tests are consistent and reproducible, not just in test inputs but in replicating the test environment

Paul has been looking at System Theoretic Process Analysis (STPA) as a way of modelling systems to provide a basis for deriving safety test cases. Not only describing how the system behaves but also a way of thinking about the negative outcomes (eg. unsafe control actions) that need to be avoided. it is also often not possible to prevent negative outcomes so ways of mitigating those outcomes need to be considered.



Once the control structure is established, consideration is then given to the impact if those actions are not provided, provided, provided too late/too early and provided too much/too little.

This can then be used to derive and refine safety tests but the process needs to be iterative, even through into live operation, as new issues emerge and to cope with changes to the system requirements.

Ensuring that stakeholders speak a common language is also important as they have different areas of expertise. Using a restricted-form natural language (RNL) can help give stakeholders a clear understanding of tests and their purpose and allows the derivation of logical tests.

Automating system-level tests has a number of considerations:

- What is being tested, how was it built, what else is part of the system and what context is it tested in?
- What actions does the test involve? What observations need to be made and what criteria need to be satisfied?
- How are result collected and published?

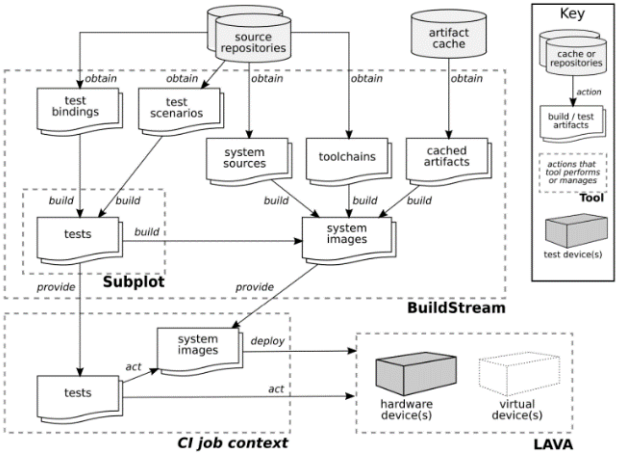
Paul introduced an example of an automation tool called **Subplot** that provides a tool for documenting and executing tests. Scenarios are written in a mark-up language that match syntactical elements (nouns, verbs etc.) to test implementations. Tests can in theory be implemented in any language with current implementations supporting Rust and Python.

Other tools discussed were:

BuildStream allows systems and tests to be built deterministically.

LAVA: manages devices (hardware and virtual) used in system tests, maintaining pools of devices with defined characteristics.

Paul concluded by saying these products can be combined in an ecosystem to automate the end-to-end process as shown in the figure.



Bootstrapping Safety Assurance

Phil Koopman from Carnegie Mellon University concluded the seminar on "bootstrapping" safety assurance using examples drawn from the area of the assurance of autonomous vehicles.

Phil said the autonomous vehicle industry started out by "testing with miles", or simply operating vehicles in the field to gather data and learn from emergent issues. This method however is expensive and there is a risk of high-profile crashes occurring. Nevertheless, several car manufacturers are using the safety of autonomous vehicles as a selling point.

Phil asked how safe is "safe" and noted that the current US fatality rate is ~100M miles per fatal mishap and noted that this includes those with alcohol impairment (28%), speed related (26%) and distracted drivers (9%). The target comparison should therefore be a fully functional driver (unimpaired, undistracted) with a modern vehicle (ie. with modern safety features).

Phil said the target safety rate for autonomy is a difficult question (entire books have been written on it) but suggested a ballpark figure of 200M miles per critical mishap. It is however, impracticable to test using this "brute force" method as it would require 2 billion miles of testing (50 round trips on every road in the world) to be statistically meaningful.

Simulation to supplement road tests is seen as a more feasible and cost alternative; it is scalable and cheaper and capital equipment outlay is much less. However, models are not perfect and there will still be a need for real-world testing to validate the simulations.

Phil introduced the Bootstrapping approach to road testing assurance, which follows the incremental steps as follows:

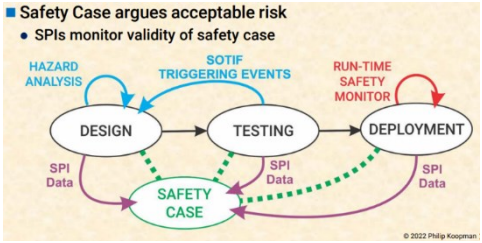
- Observe safety drivers are not intervening
- Remove the safety driver
- Crash-free history predicts crash unlikely for small window
- Drive for small window with no crash
- Repeat last two steps with growing window size

He then, through the use of an example of a successful 10-mile US truck demonstration, argued that because a one-off test did not result in any safety incidents, it does not mean the system is safe. However, the bootstrap argument being used is that every low-risk incremental test without incident contributes to the overall confidence that the system is safe.

Phil then said that if there is a crash during testing, then the bootstrapping argument means you need to test for longer and the maths is not in the tester's favour as it quickly becomes impractical, even if there are only a few crashes. Phil said companies are finding ways to discount crashes, eg. it was the other driver's fault or it was a freak/black swan that's now been fixed.

Phil argued that there is no such thing as "uncrewed AV safety testing" and is really just deployment of unproven technology.

Slow-rolling and pilot deployments also do not help as it is just a manifestation of the Bootstrap argument. They reduce the risk of multiple concurrent early mishaps but this does not mean that systems are any safer.



Phil suggested a potential approach for the issues raised, which is speculative based on Safety Performance Indicators (SPIs). An SPI is a metric supported by evidence that uses a threshold comparison to condition a safety case claim. The metrics are measurements of performance, design quality, process quality, operational procedure conformance etc. Importantly, a violation of an SPI shows that part of a safety case has been falsified rather than a failure of the vehicle. This allows SPIs to be based on aspects that don't involve crashes. A sketch of an argument would be:

- Industry safety standards and strong safety culture
- Known hazards have been mitigated
- Arrival rate of unknowns is low
- Safety has good SPI coverage
- SPI usually detect unknowns without an actual crash

This can then be used to bootstrap on surprise arrival rates and SPI improvements.

Report by Paul Hampton SCSC Newsletter Editor

Image Attribution: lead car: 130733406 / Safety Testing © Andreistanescu dreamstime.com. Other images copyright original authors.