It is common within safety critical systems to ask the question either during the design / requirement capture stages ‘is the design testable?’ More often than not the answer is given without too much consideration to what is being asked. The project is asking “Is it Testable” and “Is it straightforward to test” and “and is this a reasonable design from a testability point of view” but the person answering the questions hears “Do you think it could be tested”, in which case the answer “Yes” means “Probably” or “I don’t see why not”.

Testability of requirements and designs can be a secret and costly killer of safety critical projects both large and small. It is very easy to see projects progress nicely all the way through the development stages giving a false sense of security to the project team, only for it to then slow to a crawl during the testing phases or even worse, the design be sent back to the drawing board as it is impractical or is completely untestable.

The mantra managers of safety critical projects should recite is that regardless of how clever, brilliant or revolutionary a system design is, “if it can’t be proven / tested, then it is useless”. Resource Group has decades of experience within the domain of safety critical software testing and verification, across different developments environments, applications and customers. Over the years we have developed a clear understanding of what is a testable design and what are the pitfalls to avoid. This paper discusses some of that experience.

Requirements

It is well known that the higher up the lifecycle a mistake is made, the bigger the impact in both time and money to rectify it. From experience, poorly written / defined requirements constitute the biggest source of untestable designs. The freedom of language, in defining requirements and the variations in how different engineers will document the same need can lead to a wide difference in the testability of the design.

There are of course a few big red flag examples:

- Negative requirements – “The system shall never enter state X during normal operation”. It is impossible to test that something will not happen without very specific criteria. It should also be noted that if you have requirements stating when something will not happen, it is probably because you are missing some positive requirements covering when it shall / should happen.

- Background Information in requirements – The best requirements are those that are clear (readable), concise and complete. Having a rationale for a requirement is good, but including that rationale within the requirement is confusing and increases the testing effort to distinguish / justify what needs testing and what doesn’t.

- Identifiable inputs / outputs – A common problem considering recent EASA and FAA guidance on the use of pseudo code and reverse engineering is judging the correct levels of abstraction between each of the development stages. It can commonly result in designs that cannot be tested from only the requirements written; inputs/outputs used by the requirement also need to be known. Unless signals are detailed within the defined requirements (which can cause its own problems) a methodology is needed that will allow the testers to map input /
outputs onto the words being used in the requirements. The risk here is that testers have to analyse code / data dictionaries and make guesses using engineering judgement, at the correct signals for the requirement being tested.

- Ambiguity – The variation of words / meaning used by different engineers (or sometimes even the same engineer) to implement the same functional requirement can be significant. If a requirement is ambiguous it is the duty of the test engineer to assume the incorrect interpretation and get the design team to remove the ambiguity.

- Incomplete requirements – The designer writes the requirement defining the positive action to take but fails to indicate what action should be taken if the positive action doesn’t occur e.g. set to the opposite state or leave unchanged. The designer may feel that he does not need to define the opposite action because it is obvious if you understand the entire system. This issue occurs because the designer has a wider scope of the application and has designed top down from the customer requirements so has a better understanding of the overall system. Testing on the other hand is going bottom up, so the answer is not obvious because they have limited scope of the application and knowledge grows as more testing is performed.

The things that effect the complexity of tests are:

- Complexity – Design looks at the complexity of the code and nesting via complexity metrics like McCabe’s, but this is not a perfect indicator of effort when considering unit / module testing.

Detailed Design

The testing of detailed designs (normally through unit / module test) is not immune from testability pitfalls also. The problems encountered here tend to be due to a lack of understanding of how testing is performed rather than the detailed designs themselves:

- Complexity – Design looks at the complexity of the code and nesting via complexity metrics like McCabe’s, but this is not a perfect indicator of effort when considering unit / module testing.
• Hardware reads – Often designs for validating signals etc. start by reading hardware registers and then go on to validate the signal. In most testing environments (host testing) this hardware read cannot be controlled by the tester so he/she cannot control the data being returned and hence cannot verify the design. As a rule of thumb, put hardware reads and writes into subroutines with return parameters that can be stubbed by the tester and so the data flow can be controlled.

**Conclusion**

Designing for Testability has many possible solutions but more potential pitfalls. Ideally, if design engineers were required to do testing for a period they would start to understand the issues faced by the testers and design more testable code. A shorter term and very effective solution is to get the testing / verification team (during the development phase) to sign off on the testability of the design to ensure that any potential testing issues are resolved before the design is finalised. Both of these solutions can be aided greatly by correctly considered and enforced procedures / standards and appropriate training. As with most problems, foresight is the key, and taking the time to consider how something will be tested before putting pen to paper could save significant costs downstream.