James Buncle
878244

Testing Software with High Degrees of Complexity, Uncertainty and Novelty

1.0 Introduction

Complex, uncertain and novel software systems create a greater challenge when it comes to testing. The majority of traditional techniques are often more difficult to use to test such systems, and are often inappropriate or not thorough enough.

In commercial software, which are often considered as complex, there is always a compromise between getting the product to market and releasing it without errors. Frequently, this is because testing can be as complex as the software itself. This means that to fully test the whole system is costly and time consuming. If testing it takes too long and postpones release, the market may have moved on and there may be no longer any need for the software.

2.0 Problems with Testing

2.1 Complex Software

With complicated software it is often just as difficult to test as it is to design and build. This therefore makes testing a resource consuming process, making it no longer feasible to test the whole system completely. Therefore it becomes more sensible to prioritise functionality, and test accordingly.

Another problem with complex software testing is that any changes in the requirements of the software during the development of it will have a cascading effect on the software system and therefore on the testing of it. A small change in part of the software due to alteration to the requirements document will often have an effect on all the parts that are dependent on it. Therefore, it is important to be able to identify all these parts in order to determine the effect on the design of the tests. This itself can be a lengthy process, therefore the tests, as with the software, need to be designed with maintainability in mind.

The more complex the software, the more difficult it is to manually complete tests, therefore a need to automate testing as much as possible is needed. The automation of testing is a relatively new and currently a difficult task in itself.

2.2 Software with Uncertainty

In order to thoroughly test whether or not the software fulfils its original and intended purpose it is necessary to have a clearly defined, high quality requirements document. This is normally required to be able to design tests which cover all, or at least the majority, of possible circumstances which can occur. With software that is shows uncertainty, it is difficult to specify the exact functionality required. This therefore makes it very difficult to test thoroughly, after all, how can the software be tested if it is unclear what is to be tested?

2.3 New Software

When testing software that is similar to existing or previously designed software, older test designs and techniques can be reused as a basis for the new software. However when the software, or part of the software, is new and hasn't been done before, the design of tests becomes more difficult as this will be as new as the software itself.

3.0 Black Box and White Box Testing

In software engineering, testing is divided into two main types, Black box testing and white box testing.

Black box involves the testing of software, overall or in sections, from an external perspective whereas white box testing is based on the internal software structure and its code.

3.1 Black Box

Black box involves techniques such as specification-based testing, which is used to validate whether the software fulfils the software requirements, verifying whether or not the software does what it is supposed to. This technique is dependent on a good quality specification, which for new and complex software is difficult to do.

For software that has a high degree of uncertainty, black box testing is generally more difficult, as a high quality specification is difficult to produce. The poorer the quality of the specification, the more difficult it is to design the black box tests.

The two main black box techniques are acceptance testing and system testing:

3.1.1 Acceptance Testing

Acceptance testing is completed by the sponsor who is then able to verify whether or not the system meets the criteria requested.

Typically this involves completing various tests, known as a test suite, on the completed system. Each test, known as a test case, has a boolean pass or fail result.

The tests are designed with the aim to be identical to or as close as possible to real world usage.

If all cases in a test suite are correct then the suite is considered as a success.

This method of testing may be more appropriate with software that has a high degree of uncertainty , complexity and novelty as it allows the sponsor to judge whether or not the software does what they originally intended. Ideally, this would be better used on an iterative basis, where the software is demonstrated to the sponsors regularly on a prototype basis. This should reduce the amount of work that would otherwise be wasted if the sponsors were only given the pre-final version, which would only then be rejected and need to be redone. The problems arise with designing the tests, which would generally need to be initially vague, however over time the tests can be developed and therefore become more robust.

3.1.2 System Testing

System testing is used to verify that the entire software system satisfies the specified requirements. It is completed after the system has been integrated and it used to evaluate it as a whole. This method involves the implementation various techniques to test the functional requirements of the system.

For software that is uncertain, where possible this will give a more definitive method of determining whether the software does what it is supposed to. Generally, there will be an overall understanding of what is required, but the details uncertain. As this is an overall evaluation of the software, it seems more appropriate. However, if any of the tests fail, the causes of the problems will be difficult to identify.

3.2 White Box Testing

White box testing includes techniques that involve manipulating and testing the code directly. An example technique is fault injection, where a fault is deliberately introduced into the code so that the effects can be observed. Other techniques involve testing individual parts of the software, such as classes. This is useful for complex software as there is an opportunity to break it down into smaller, simpler parts, making it easier to test.

The two more popular types of white box testing are unit and regression testing:

3.2.1 Unit Testing

Unit testing involves testing small units, typically a single class, separately from the rest of the system. The units are normally the smallest testable parts that the system can be broken down into. Each individual test is known as a test case, and is normally independent from other parts of the system. The interaction of the part being testing with the rest of the system is simulated by specifying parameters.

When using this method to test complex software it can be difficult to foresee all possible interactions between units. It may be found that every unit passes the tests designed, but will not work correctly when fully integrated.

This method is suitable for software that has a good quality requirements specification as the individual tests can be designed based on these. However where the software is uncertain is can be difficult to understand what each unit is meant to do, therefore increasing difficulty in designing the tests.

3.2.2 Regression Testing

Regression testing is used to evaluate the functionality that is new or has been developed in the software. This is done be rerunning tests designed for the original or older code to determine whether any old functionality is lost. This may involve retesting the whole system or selecting appropriate parts to retest using Regression Test Selection (RTS).

Again, this method is useful for complex software, as it saves time through the reuse of tests. This means that tests are not redesigned unnecessarily and can simply be developed with the software changes. This is relatively straightforward for testing the independent parts in the software or the software overall, however it can be a time consuming process, as it will often involve retesting parts that are unaffected by the changes. This is where RTS becomes important, which involves selecting appropriate parts.

The test designers will therefore need to determine whether or not to do an overall test of the section involved or to use RTS.

4.0 Verification and Validation of Complex Algorithms and Mathematical Models

Complex algorithms and mathematical models are developed for a specific purpose. The validity of these is determined by their ability to fulfil that purpose correctly. In order to verify the success of these, the algorithms need to be tested using several different sets of parameters and conditions. This is done because an algorithm may be valid for one set of conditions, but invalid for another.

Often it is too expensive on resources to try to determine absolute validity, therefore testing will often be performed until a certain confidence is reached and the algorithm is considered valid enough for its purpose.

4.1 General Approaches

There are three main general approaches to the validation of software models and algorithms, each approach can use various techniques to determine validity.

The first and most popular approach is where the developers of the model determine whether or not the code is valid themselves. The developers make a subjective decision based on various evaluations and tests conducted previously.

Another approach is to use 3rd party testers to determine validity, this method is known as Independent Verification and Validation (IV& V). The 3rd party normally consists of individuals that are independent of both the developers and the sponsors of the algorithms. Their purpose involves performing various evaluations of the algorithm to determine its validity. This method is costly and is therefore normally only used if the development of the model has a high amount of capital invested in it.

The third and least popular method is to use scoring model. This involves breaking down the algorithm into separate parts which are tested individually using scores or weightings. The algorithm is then evaluated by taking an overall score. This is then compared to a predetermined pass score to determine validity. If the collected value is over the predetermined score then the algorithm is considered as valid.

4.2 Techniques

There are a wide range of techniques that can be used within the approaches specified above to verify models. A few examples are suggested below.

One technique is to display the algorithms graphically, this can then be observed over time and judgements can be made as to whether or not the algorithms are operating as expected. This method is very dependent on the type of algorithm being represented, as displaying some algorithms in a graphical way may be inappropriate. It also depends on the quality of the representation, as if it is visually unclear it will be difficult to evaluate.

Another technique is to use Degenerate Tests, where portions of the model are removed (or replaced) or internal parameters are altered. The algorithm is then tested to see how it copes with the new modifications. This is also a good method to test the robustness of the algorithm, if it handles well with the modifications it can be considered as more robust than if it didn't.

Extreme-Condition Tests involve testing for extreme or unlikely circumstances. For instance by using inputs that should not, or are very unlikely to occur. This method of testing is useful for algorithms designed to be used with highly varying parameters, however it is often difficult to know exactly what the extremes are. It is also possible to get overly confident, it may work for rare circumstances, but not for common ones.

Another option is to control the values, so that single parameters are tested separately, one at a time. This involves fixing input and internal values as constants, to create a controlled environment. This is probably the most scientific approach to test. Another technique is the opposite of this, where various values are changed and the effect on the model and outputs are evaluated, this is known as Parameter Variability.

5.0 Conclusion

The testing of complex software manually is a lengthy and resource consuming process. Therefore the future will see an increase in the use of automated testing. This automation may be code based or graphically based. The code based automation generally involves white box testing, where classes or packages are tested from within the code. The problem with this however, is that the testing may have an effect the codes environment. The solution to this is to submit the code with the test code within it.

The other automation option is through the use of a Graphical User Interface, allowing the user to execute tests which are partially automated.

Each approach to testing has its problems, therefore it is generally necessary to use a combination of testing approaches, ideally one test would then compensate the problems of another. However as mentioned previously, with software that is complex, uncertain and novel, some approaches are not appropriate.

As testing software can be a time consuming process, there is often compromise between releasing a buggy system, and releasing it early. Therefore, testing has to be prioritised to reduce the impact that problems may have on the system. The idea being that the crucial parts of the system will be tested more thoroughly than less important elements of the system.

Bibliography

[1] Verifying and validating simulation models

[2] Jan Stafford, “Testing Strategies for complex environments”, TechTarget, 2009

[3] Erman Coskun, Martha Grabowski, “Software complexity and its impacts in embedded intelligent real-time systems”, Journal of Systems and Software

[4] Marc-Philippe Huget, “Executing Ultra-Large Software Systems with MultiagentSystems”, Universite de Savoie, 2008

[5] V. V. Lipaev, “A Methodologyof Verification and Testing of Large Software Systems” Moscow 109004 Russia, 2003

[6] Alessandro Orso, Nanjuan Shi, and Mary Jean Harrold, “Scaling Regression Testing to Large Software Systems”, Georgia Institute of Technology, 2004

[7] Winston W. Royce, “Managing the Development of Large Software Systems”, The Institute of Electrical and Electronics Engineers, 1970

[8] Jiantao Pan, “Software Testing”, Carnegie Mellon University, 1999

[9] Sagar Naik, Piyu Tripathy, “Software Testing and Quality Assurance: Theory and Practice”, Wiley InterScience, 2008

[10] Applabs, “Future of SoftwareTesting”, 2008

[11] James Bach, “Risk and Requirements-Based Testing” 1999

[12] Laurie Williams and Sarah Heckman, “Software Engineering: Testing”, OpenSiminar, 2008