System Testing

Stuart Anderson

School of informatics

Overview

- System testing is very heterogeneous and what we include in the system test will depend on the particular application.

- The following is a list of kinds of tests we might consider applying and some assessment of their strengths and weaknesses.

- System testing can be very expensive and time consuming and can involve the construction of physical components and software to provide the test environment that may exceed the cost of the primary software development.

Capacity Testing

- **When:** systems that are intended to cope with high volumes of data should have their limits tested and we should consider how they fail when capacity is exceeded
- **What/How:** usually we will construct a harness that is capable of generating a very large volume of simulated data that will test the capacity of the system or use existing records
- **Why:** we are concerned to ensure that the system is fit for purpose say ensuring that a medical records system can cope with records for all people in the UK (for example)
- **Strengths:** provides some confidence the system is capable of handling high capacity
- **Weaknesses:** simulated data can be unrepresentative; can be difficult to create representative tests; can take a long time to run

Stress Testing

- **When:** in systems that are intended to react in real-time e.g. control systems, embedded systems, e.g. anti-lock brake system
- **What/How:** usually we are interested in bursty traffic and environmental extremes (e.g. lots of sensor messages together with cold and wet conditions). Could build a test rig to test the integration of the mechanical, electronics and software. If changes are required in service then we can use a real system.
- **Why:** these systems are most depended on in these kinds of conditions - it is imperative that they function here.
- **Strengths:** this is an essential kind of testing for this class of systems unavoidable.
- **Weaknesses:** test environment may be unrepresentative or may omit a component (e.g. in a car the radio environment is very noisy we may need to simulate this).

Usability Testing

- **When:** where the system has a significant user interface and it is important to avoid user error — e.g. this could be a critical application e.g. cockpit design in an aircraft or a consumer product that we want to be an enjoyable system to use or we might be considering efficiency (e.g. call-centre software).
- **What/How:** we could construct a simulator in the case of embedded systems or we could just have many users try the system in a controlled environment. We need to structure the test with clear objectives (e.g. to reduce decision time,...) and have good means of collecting and analysing data.
- **Why:** there may be safety issues, we may want to produce something more useable than competitors' products...
- **Strengths:** in well-defined contexts this can provide very good feedback – often underpinned by some theory e.g. estimates of cognitive load.
- **Weaknesses:** some usability requirements are hard to express and to test, it is possible to test extensively and then not know what to do with the data.

Security Testing

- **When:** most systems that are open to the outside world and have a function that should not be disrupted require some kind of security test. Usually we are concerned to thwart malicious users.
- **What/How:** there are a range of approaches. One is to use league tables of bugs/errors to check and review the code (e.g. SANS top twenty-five security-related programming errors). We might also form a team that attempts to break/break into the system.
- **Why:** some systems are essential and need to keep running, e.g. the telephone system, some systems need to be secure to maintain reputation.
- **Strengths:** this is the best approach we have most of the effort should go into design and the use of known secure components.
- **Weaknesses:** we only cover known ways in using checklists and we do not take account of novelty using a team to try to break does introduce this.

Performance Testing

- **When:** many systems are required to meet performance targets laid down in a service level agreement (e.g. does your ISP give you 2Mb/s download?).
- **What/How:** there are two approaches - modelling/simulation, and direct test in a simulated environment (or in the real environment).
- **Why:** often a company charges for a particular level of service - this may be disputed if the company fails to deliver. E.g. the VISA payments system guarantees 5s authorisation time delivers faster and has low variance. Customers would be unhappy with less.
- **Strengths:** can provide good evidence of the performance of the system, modelling can identify bottlenecks and problems.
- **Weaknesses:** issues with how representative tests are.

Reliability Testing

- **When:** we may want to guarantee some system will only fail very infrequently (e.g. nuclear power control software we might claim no more than one failure in 10,000 hours of operation). This is particularly important in telecommunications.
- **What/How:** we need to create a representative test set and gather enough information to support a statistical claim (system structured modelling supports demonstrating how overall failure rate relates to component failure rate).
- **Why:** we often need to make guarantees about reliability in order to satisfy a regulator or we might know that the market leader has a certain reliability that the market expects.
- **Strengths:** if the test data is representative this can make accurate predictions.
- **Weaknesses:** we need a lot of data for high-reliability systems, it is easy to be optimistic.

Compliance Testing

- **When:** we are selling into a regulated market and to sell we need to show compliance. E.g. if we have a C compiler we should be able to show it correctly compiles ANSI C.
- **What/How:** often there will be standardised test sets that constitute good coverage of the behaviour of the system (e.g. a set of C programs, and the results of running them).
- **Why:** we can identify the problem areas and create tests to check that set of conditions.
- **Strengths:** regulation shares the cost of tests across many organisations so we can develop a very capable test set.
- **Weaknesses:** there is a tendency for software producers to orient towards the compliance test set and do much worse on things outside the compliance test set.

Availability/Reparability Testing

- **When:** we are interested in avoiding long down times we are interested in how often failure occurs and how long it takes to get going again. Usually this is in the context of a service supplier and this is a Key Performance Indicator.
- **What/How:** similar to reliability testing – but here we might seed errors or cause component failures and see how long they take to fix or how soon the system can return once a component is repaired.
- **Why:** in providing a critical service we may not want long interruptions (e.g. 999 service).
- **Strengths:** similar to reliability.
- **Weaknesses:** similar to reliability – in the field it may be much faster to fix common problems because of learning.

Documentation Testing

- **When:** most systems that have documentation should have it tested and should be tested against the real system. Some systems embed test cases in the documentation and using the doc tests is an essential part of a new release.
- **What/How:** test set is maintained that verifies the doc set matches the system behaviour. Could also just get someone to do the tutorial and point out the errors.
- **Why:** the user gets really confused if the system does not conform to the documentation.
- **Strengths:** ensures consistency.
- **Weaknesses:** not particularly good on checking consistency of narrative rather than examples.

Configuration Testing

- **When:** throughout the life of the system when the software or hardware configuration is altered.
- **What/How:** usually we record a set of relationships or constraints between different components e.g. libraries and systems, hardware and drivers and if we change any component we should check the configuration is maintained.
- **Why:** configuration faults will often cause failures if they are not corrected.
- **Strengths:** addresses an increasingly common source of error. It is possible to automate this style of testing.
- **Weaknesses:** only as good as the record of dependencies recorded in the system. There is no universal approach but there are emerging standards to record configurations.

Summary

- There are a very wide range of potential tests that should be applied to a system.
- Not all systems require all tests.
- Managing the test sets and when they should be applied is a very complex task.
- The quality of test sets is critical to the quality of a running implementation.