VERIFYING SOFTWARE ROBUSTNESS

Ross Collard

Collard & Company

OVERVIEW

- Software is robust if it can tolerate such problems as unanticipated events, invalid inputs, corrupted internally stored data, improper uses by system operators, unavailable databases, stress overloads and so on.
- Systems that include both hardware and software are robust if they can tolerate physical problems such as equipment damage, loss of power, software crashes and so on.
- Since these problems can and do occur in live operation, this session examines how to evaluate a system’s robustness within the relative sanctity of the test lab.

AGENDA

Basic definitions

Testing for robustness

Evaluating reliability

Common complications

Risk-based testing

BASIC DEFINITIONS

What is Robustness?

- *Robustness* is the ability of a system to prevent, detect, adapt to and recover from operational problems.

What is Robustness Testing?

- Since these problems can and do occur in live operation, it is important to evaluate a system's ability to handle them. *Robustness testing* tries to make a system fail, so we can observe what happens and whether it recovers.

BASIC DEFINITIONS

- *Reliability* is most commonly defined as the mean time between failure (MTBF) of a system in operation, and as such it is related to availability.

- *Scalability* is the ability of a system to accommodate increases in work load, number of users, database size, etc.

BASIC DEFINITIONS

- A *stress test* is one which deliberately stresses a system by pushing it *beyond* its specified limits.
- *Recoverability* is the ability of a system to return to operation after a failure.

SOME COMPLICATIONS

- Systems, especially complex ones, often behave in ways which their designers neither anticipated nor understand.
- Systems can fail in many different ways.
- When systems fail, the diagnostic audit trail is often incomplete.
- Little failure data is publicly available which we can use to guide the testing efforts and fault injection.

SOME COMPLICATIONS

- Many developers and testers are unaware of techniques developed by the robustness and dependability community.
- The testing tools themselves can fail or act bizarrely under stress.
- We do not know what work loads to test with.

SOME COMPLICATIONS

- All components of a system affect its dependability, so evaluating end-to-end dependability needs a multi-disciplinary approach.

- The test environment does not mimic the live environment. Etc., etc.

TYPES OF ROBUSTNESS TESTING

A. Violations of Pre-Conditions

B. Heavy Loads

C. Probing for System Limits

A. Violations of Pre-Conditions

Negative testing

- Invalid inputs

Boundary value testing

- Limits of ranges; edge conditions

De-stabilization

- Mutation analysis and perturbations

B. Heavy Loads

Load testing

- Heavy and peak loads

Limit testing

- Testing at specified limits (often by contract)

Stress testing

- Overloads

Hot spot testing

- Intense, narrowly focused loads

C. Probing for System Limits

Bottleneck identification

- Uses invasive probes to monitor resource use

Duration or endurance testing

- Long-fuse, delayed action failures, e.g., memory leaks; 24- to 96-hour durations

Accelerated life testing

Enriched failure opportunities; shortened duration

Spike and bounce testing

- Intense sudden surges of demand; simulation of volatile conditions

Breakpoint testing

- Increase load until system fails; find the breaking point

D. Interactions

Rendezvous testing

- Coordinate multiple concurrent events

Synchronization testing

- Timing, sequence of events, race conditions

Feature interaction testing

- Interference testing

Deadlock testing

- Database contention, contention for latches

E. Human Errors

Bad day testing

- Operator and user flubs

Soap opera testing

- Exaggerated user scenarios

F. Catastrophes

Disaster recovery testing

- Identification of disaster scenarios
- Disaster recovery plan lends credence to implausible scenarios
  - – Nasdaq multi-user log-on failure

G. Physical Failures

Environmental testing

- Physical conditions – temperature, electricity, radiation, pollutants, vibration, G forces (gravity), etc.

H. Handling Changes

Live change testing

- Make modifications while running live

Invalid configurations

Change to unsupported settings

Use extreme corner cases

J. Handling Errors

Error detection & recovery testing

- Reverse engineering from error messages

Degraded mode testing

- Run with some facilities disabled

Software fault injection

Triggers inserted to cause system failures deliberately, in test mode

SOFTWARE FAULT INJECTION

- *Software fault injection* is a specialized type of design for testability, to provide the testers with the capability to easily, safely trigger or simulate system errors which otherwise might be very difficult to observe in the test lab but which nevertheless may happen in the real world.

- Software fault injection is different from software fault insertion, which is a way of assessing test effectiveness by deliberately inserting errors into systems in an experimental mode.

MODES OF FAILURE

- One of the main objectives of a stress or robustness test is to see if we can make the system fail within the relatively safe and controlled confines of the test lab, in order to observe the conditions under which the system fails, how it fails (what happens), and whether it recovers in an acceptable manner.

- Many people believe that a system can only fail in one way or at most a small number of ways. They also believe that, in any case, the different ways in which the system could fail are not very important to the users (and the testers).

AUTOMATED ROBUSTNESS TESTING

- Automated robustness testing has the advantages of being more comprehensive, cheap and fast, but it tends not to have the same degree of creative destruction as a devious human tester.

- Example: Phil Koopman of Carnegie Mellon University (CMU) built a tool called Ballista to test operating systems. Ballista generates test cases, using combinations of valid and invalid (positive and negative) inputs.

RELIABILITY

- The probability of executing for a period of time without failure (MBTF).
- Measured reliability depends on the failure model, load and infrastructure.
- Not the same as availability, recoverability, robustness (but related).

SOFTWARE RELIABILITY ENGINEERING (SRE)

The intention of SRE is to answer two questions:

- (1) Given the pattern of failures found in system testing, what level of system reliability can we realistically expect to experience in live operation?
- (2) If a goal has been set for a system’s reliability in live operation, when can we stop testing the system and removing defects, because the system has become clean enough to meet the goal?

SOFTWARE RELIABILITY ENGINEERING

Limitations of SRE

- The SRE method requires a large number of data points (i.e., failures incidents) to work, like any statistics-based method.
- The method is only as good as the operational profile which is used – it needs to match reality
- The method is only as good as the reliability estimation model.
- The reliability estimates are based on extrapolations from past experience.
- Test coverage is likely to be low, since the distribution of the test cases adheres to the operational profile.

TEST DURATION

- How long do we need to execute the software in order to accumulate enough failure data? One way to answer this is by trial and error -- keep counting until we have accumulated enough data. Of course, this means that the testers cannot give any estimate of the testing duration until after it is completed.

- According to John Adams of IBM, most software defects result in failures only rarely. The average software defect found after delivery in large systems has an MTBF (mean time between failures) of 900 years, and 35% of defects have an MTBF greater than 5,000 years.

SOFTWARE ENTROPY

- Software entropy, also called software rot, is the phenomenon by which software reliability decreases gradually over time, because of the propensity of patches to introduce inadvertent new defects. Even if the software is not modified, it becomes obsolete because the world continues to change around it, so its reliability degrades regardless.

- After a certain age (usually anywhere from 2 to 5 years after the system was first implemented), the rate at which new defects are introduced through modifications exceeds the rate at which they are being removed.

THE SCALABILITY ISSUE

- Scalability is the capability of a system to expand (or contract) as the needs change, and to provide acceptable service as the load increases or decreases: i.e., to handle large as well as small loads, large as well as small databases, and large as well as small networks.

- Scalability problems can happen with new systems which have been expressly designed for growth, but tend to be worse with existing infrastructures which have evolved.

CAUSES OF BOTTLENECKS

- Imbalances
- Data Capacity Limitations
- I/O and Bandwidth Capacity Limitations
- Processor Limitations