Climbing Mount Reliability

Senior Consultant at nFocus Sam Clarke describes the approach used by nFocus to improve and maintain reliability through and beyond the project development lifecycle.

Everyone becomes a 'test expert' as time and budget pressures take over, resulting in poor quality of testing and poor reliability.

There is incessant pressure to supply reliable systems at reduced cost and increased profitability. Object-oriented development has also made it easier to develop components iteratively in isolation, for example database access procedures can be developed independently from the application or the user interface.

Traditional testing methods attempt to improve reliability by debugging the system using a combination of reviews and unit testing followed by independent system and integration testing.

The later stages of verification and validation testing (acceptance and pilot programmes) should continue to improve reliability until the system is deemed reliable enough to be implemented. This applies whatever type of development lifecycle – waterfall, RAD etc. – is used.

How do we measure reliability?

There are many indicators of reliability. Some are listed below but the supplier, client or end-user should determine what is meaningful for the system under development. A good start can often be made by considering what criteria are used to measure the reliability of an existing system. One common thread throughout is the need for some ratio or reliability index based on problems found, so problem or incident collection and management is a key requirement for reliability measurement. My advice is to find as many measures as possible, plot them over time and look for a trend. These measurements could well be used as part of the acceptance criteria.

Some potential measurements to collect over fixed time periods:

- Tests executed per problem;
- Regression errors found;
- Time testing per problem;
- Database accesses per problem;
- Processor cycles used per problem.

Use these indicators with caution; we are looking for trends and small samples can give misleading results.

What often happens

During system development the schedule is affected by changes; examples include user requirements, technology, design, user interface, interaction with other systems and changes to deadlines.

This results in pressure from project personnel to compromise the agreed test process. Often test stages are overlapped, testing is deferred into later test stages (which were never designed for that purpose) and worst of all some testing is just abandoned. Everyone becomes a 'test expert' as time and budget pressures take over, resulting in poor quality of testing and poor reliability. The end result is often an unreliable system that is incomplete, doesn't meet specification, doesn't perform to expectations, has poor security and is difficult to manage and maintain.

As testers we have to accept that changes will occur whether we like it or not. Trying to impose strict discipline in the early stages of development only alienates the rest of the project disciplines, so why not accept the inevitable and, as a team, plan for this change? However there will be some rigour required in the process if we are to maintain and improve reliability, particularly toward the later stages of testing where any change can have serious consequences.

How could we solve these problems?

If we accept that this situation is going to arise then we should have a test strategy that assumes that change is inevitable. It should define a change process that allows the plan to be flexible enough to cope with unforeseen changes that could adversely affect reliability.

The key to reliability improvement lies in the code and unit test stage of development.

What if we could improve by continually testing and retesting the system components as they are developed?

The effect of change could be seen and dealt with immediately, and errors could be fixed early, reducing the risk of poor reliability resulting from fixing the backlog of errors late in the project cycle. Regression failures would immediately show any process failure or unexpected effects on the system. This maintains the reliability of the components developed so far and should improve the system reliability as a whole as further components are developed and integrated together.

The cumulative set of regression tests would continually show that the components delivered to system test are at a known level of reliability and integrate together.

This solution cannot be achieved using conventional manual testing due to cost, time and effort. The answer lies in the automation of the testing. However using record and playback tools in a conventional manner relies on a number of factors:

- The system must be complete and stable enough to allow the tool to drive the tests;
- Test analysts will need in-depth knowledge of the tool set;
- The scripts need continual maintenance;
- The tools tend not to be able to drive existing application or database harnesses.

A solution

If we can abstract the creation of the automated tests from the test input, the restrictions stated above are significantly reduced. Figure 1 shows how this process works. We should consider the system under test as consisting of areas of functionality: each area consists of separately developed testable components, which integrate together to deliver the business function. These areas comprise user interface, application, transport, database and batch processes.

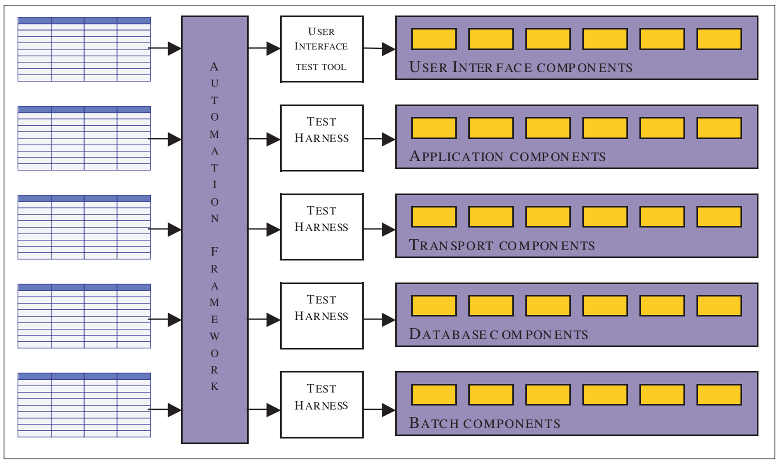

Figure 1.

The tests are abstracted away from the tool set (harness or record and playback tool) to a standard, understandable and easy-to-use form such as a spreadsheet.

Each area can be subdivided into components that can be developed and tested independently, e.g. a Web page, a Web service or method, a database stored procedure etc. Eventually enough components can be integrated together to deliver a piece of business functionality.

The tests are abstracted away from the tool set (harness or record and playback tool) to a standard, understandable and easy-to-use form such as a spreadsheet. The spreadsheets can then be used by an automation framework to generate scripts dynamically for whatever tool or harness that is in use.

The entries in the spreadsheet become the test assets, with the automation scripts being discarded after each execution. This allows harnesses and toolsets to be replaced or enhanced as necessary without affecting the tests, the only changes required being to the generation process used by the automation framework.

The order of component delivery is not critical as each set of tests will be built at the same time as the component is developed, however the delivered component must be put under change control after delivery. Although this may seem rigid, it is in fact quite an agile process, providing the test analysts and developers have a good rapport and the change and quality process is effective.

Our experience is that the development organisation tends to be reticent about the approach until they see the benefits of the overnight 'best so far' build test running clean or, if failing, immediately being able to pinpoint the error.

The process

A typical instance of the iterative process is outlined below:

- Agree the areas and components to be delivered;
- Develop and white box unit test component;
- Deliver to the test rig;
- Run the automated regression tests;
- Report any regression errors;
- Agree if the error is severe enough to reject the build;
- Retest any outstanding errors from the last cycle;
- Run the automated black box tests for the component(s);
- Report any problems;
- Add successful tests to the regression test set ready for the next iteration.

On larger projects you may need to select a subset, for example build verification and execute the full set overnight or over a weekend.

As more components are delivered it becomes possible to string together self-contained business process tests and demonstrate reliability improvement.

Additional benefits

Better use of scarce test resource:

- Tests are kept in a standard form enabling efficient use of test resource across projects;
- Tool independence;
- Reuse of existing harnesses and drivers by the test team;
- Tests are self-documenting;
- Best use of technically-skilled resource.

The automated tests can be used for other purposes:

- Verification of system installation;
- Demonstrating the system is working to specification;
- Showing changes to the production system or infrastructure have not had an adverse effect;
- Showing that future changes to the system have not had an adverse effect.

Conclusion

The ability to continually execute a set of automated regression tests is the key to maintaining and improving reliability while preventing the over-run of the later stages of testing due to increased debugging effort. The approach of using an automation framework frees up the tester to concentrate on testing. In addition problem determination time is reduced as any failure can be quickly isolated to a component. ■

For further information, please contact nFocus on tel: 0870 242 6235, email: info@nfocus.co.uk, or visit www.nfocus.co.uk