



# Abstract art

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

*There is incessant pressure to reduce cost and increase profitability whilst reducing the risk of project failure by supplying reliable systems.* In addition the advent of object oriented development has made it easier to develop and redevelop components iteratively in isolation: for example database access procedures can be developed independently from application methods. The application can be developed in isolation from the user interface.

Traditional testing methods attempt to improve reliability by a combination of reviews, inspections and unit testing during the design and coding stages of development. After completion of the development of the system some level of independent integration and system testing is performed to debug the system as a whole.

The later stages of verification and validation testing (user and operational acceptance and pilot programmes) should continue to improve reliability in a business and operations oriented environment 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 possible indicators of reliability. Some are listed below but you and your client or end user should determine what is meaningful for your project. A good start can often be made by considering what criteria are used to measure, objectively or subjectively, 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. Don't forget to involve the user; 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 reads/writes/updates per problem
- disk I/Os per problem
- processor cycles used per problem
- number of complaints by testers per time period

Don't forget the old "lies, damned lies and statistics" adage; use these indicators with caution. Remember we are looking for trends and small samples can give misleading results.

## What often happens!

Whether we like it or not things change in any system development project. Development of computer systems is a creative process and often the user does not fully understand the requirements for the system until after design and development has started. Some other examples of areas that can generate change are technology and design issues, performance, security, user interface, interaction with other development projects, legislation and changes to deadlines.

The result of these changes is usually that the test team is put under pressure from other project personnel (who agreed the original plan) to compromise the well-ordered and well-defined 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 with little regard for the consequences. All of this adversely affects reliability and as the test team tries to reimpose order in the resulting sea of chaos all the carefully thought out strategy and plans can be discarded and replaced with a chaotic 'heroic' plan. Everyone becomes a 'test expert' as time and budget pressures take over.

This can result in on-the-fly re-planning which often compromises the quality of testing, and consequently impacts the reliability model described earlier. The end result is often an unreliable system that is incomplete, doesn't meet specification, doesn't perform to expectations, has poor security, and/or 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 discipline required in the process if we are to maintain and improve reliability, particularly toward the later stages of system testing where any change can have serious results.

## How could we solve these problems?

If we accept that this situation is going to arise then we should have a test strategy and plan that assumes that change is inevitable. The strategy should define a change process that allows the plan to be flexible enough to cope with unforeseen and as yet unknown changes that could adversely affect reliability. The key to reliability improvement lies in the code and unit test stage of development as it is this stage of the project that is most likely to overrun.

What if we could enable and maintain reliability from the first and subsequent delivery of a component through to system test by continually testing and regression testing the system components as they are developed?

The effect of change could be seen and dealt with immediately rather than problems being found late in the project cycle. Errors could be fixed early, reducing the risk of reliability reduction due to the changes caused by fixing the backlog of errors late in the project cycle. Regression failures would immediately show any process failure (eg a failure in change control) or unexpected adverse effects on the system. This ought to maintain the reliability of the components developed so far and improve the system reliability as whole as further components are developed and integrated together.

In addition the cumulative set of regression tests would show that the components delivered to system test are at a known level of reliability and integrate together (all tests run clean) and could be reused throughout system test and beyond to show effects of any change on the system as a whole.

This process cannot be done using conventional manual testing as the cost in time and

## Understanding new technologies is our business

Established in 2000, nFocus has gained a first class reputation by delivering innovative and effective software testing solutions across many business sectors.

Our refreshing partnership culture has proved successful and we strive to provide 'real' value in a challenging and competitive industry. We are a full-life cycle test solutions company delivering consultancy, tools and unique automation techniques to a variety of blue chip clients.

## Leaders in testing the latest Microsoft technologies

As one of the UK's leading specialists in testing the latest Microsoft technologies, nFocus has the experience, processes and techniques to help you deliver successful .NET development projects.

### Creating partnerships to achieve success

We are constantly looking for experienced test professionals to join our expanding company.

People seeking a rewarding career or associates looking to cement a strategic partnership, should their send CV or services profile to [info@nfocus.co.uk](mailto:info@nfocus.co.uk)

effort would increase quickly, as the components are delivered, to a point where it would be extremely costly or impractical.

The answer lies in 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 enough to allow the tool to drive the tests; if the user interface is not stable then there is no easy way to drive the system
- most test analysts will need in-depth knowledge of the tool set
- it is difficult to scale up operations (training paths, skills etc)
- maintenance of the scripts can be problematic and take valuable tester resource away from what they do best which is testing
- the tools tend not to be able to drive existing application or database harnesses directly.

### A solution

If we can abstract the creation of the automated tests from the test input, the restrictions stated above are significantly reduced.

To explain 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 comprise user interface, transport, application, database and batch processes.

Each of these areas can be subdivided into discrete testable components which can be developed independently from the rest of the system; for example a web page, a Microsoft Biztalk orchestration, a web service or method, a database stored procedure etc. Each component can be integrated together to deliver a piece of business functionality (the normal sequence for a conventional automated test).

This allows tests to be built by existing test resources; the tests themselves are abstracted away from the tool set, whether it is a harness, or a proprietary record and playback tool, to an understandable easy to use form. This abstraction can be delivered by an automation framework which allows the user to construct the tests using a standard, simple to use interface such as a spreadsheet. The scripts for the tests can then be constructed by an automation framework for whatever tool or harness that is in use.

The entries in the spreadsheet then become the test assets, with the automation toolset scripts being discarded after each execution. This allows harnesses and toolsets to be replaced/enhanced as necessary without affecting the tests. The only changes required

are to the wrapper or to the generation process used by the framework tool.

To ensure the entire process works we also need an integrated development and test process suitably modified to have the necessary discipline to develop, deliver and put under change control the defined components. The order of component delivery is not critical as each set of tests will be built at the same time as the component.

Although this may seem rigid it is in fact it is quite an agile process providing the test analysts and the developers have a good rapport and provided the change and quality process is effectively managed.

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 pinpointing the error.

### The process

A typical instance of the iterative development process is outlined below:

- at design time, develop and agree the areas and components to be delivered
- develop a component or components and after satisfactory white box unit testing deliver to the test rig using the build process
- run the automated regression tests for that component (not first time round unless there is an existing set of regression tests)
- report any regression errors found
- for each regression error, determine if the error is severe enough to reject the build and if so reject it!
- retest any outstanding errors from the last test cycle
- update the regression set with the new successful tests
- run the 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 the regression set may well be too big to run in the allotted time in which case you will need to select a subset for, say a build verification, and execute the full set overnight or over a weekend.

As components are delivered it also becomes possible to develop simple self-contained business process tests. These tests are invaluable when moving to system test as they can give the installed system a clean bill of health prior to starting the traditional system test. Other benefits include:

- a set of automated tests. If they are signed off by the user they can be used as part of user acceptance, preventing debate over whether the system is working to specification or not.
- the ability to show changes to the production system due to upgrades of operating system, infrastructure or the application have not had an adverse effect
- the ability to show that the functionality of a future release of the system has not had any unforeseen adverse effects.

In conclusion the ability to build quickly and run continually the regression tests is the key to maintaining and improving reliability while preventing the overrun of later stages of testing due to increased debugging effort.

The approach of using an automation framework frees up the tester to concentrate on testing, adding value by developing better and more varied tests. In addition problem determination time is reduced as any failure can be quickly isolated to a component.

Key points for success are:

- the development and test process must be agreed and followed
- there must be some technical expertise to handle the development of the test harness scripts and technical issues with the test tools (though this is greatly reduced by the use of an automation framework)
- a serious regression problem should stop delivery until the cause is established and development and test agree to move forward with the error (or preferably not)
- errors in the component are fixed as soon as possible.

Our experience gained over a number of projects has shown there are potential pitfalls in the approach described here but that the overall gain in reliability and confidence is highly significant. Developers who are sceptical at first often go on to become enthusiastic. It should also be noted that this approach can't test everything; it is geared toward functional testing in the code and unit test timeframe, but delivers some significant advantages which are enjoyed in the later stages of testing.

Finally, we always suggest that an independent review of the development and test organisation and its processes should be conducted prior to embarking on a project; culture change is the single greatest source of risk of failure. 