The pursuit of excellence

Software is traditionally designed, coded, then tested. However, poor quality coding and a testing phase at the end of the process can add a significant amount of time and risk to a project.

Delayed and extended testing has a knock-on effect – the longer bugs remain undiscovered, the more likely that developers will be working with poor quality software, causing more delays as more bugs are found.

The ideal position is to have a process that supports testing as early in the development lifecycle as possible and which enables changes to be made quickly. Sounds like nirvana? It is possible.

Continuous integration (CI) focuses on the ability to build and test an application every time a change has been, or needs to be, made. Manual testing works well with a small code base, but with embedded software at the heart of so many products, this process needs to be automated to cope with the scale of the challenge.

Solving the problem of software quality and time to market is an ongoing fight, but CI helps developers and engineers address these issues.

There are five elements in the ultimate testing environment:
- Tools that allow developers to test whenever they need.
- Tools that provide visibility of testing completeness and auto generate test cases for incomplete code snippets.
- A repository that automates the job scheduling of the integration process.
- Parallelising and scaling the test architecture to achieve fastest build time.
- Overlaying intelligence that understands the smallest number of retests required by a change to the source code.

Shift left testing
The ultimate CI engine enables testing early and often. In embedded applications with millions of lines of code, leaving testing until last is dangerous; developers may need to abandon the project, while the company might face severe financial difficulty and possible penalty after discovering bugs too late.

CI is intended to be combined with automated unit testing. Historically, it was thought best to run all unit tests in the developer’s local environment and to confirm these tests were passed before committing code to the repository, avoid ‘spreading’ broken code. As CI practices have developed, the concept of build servers has appeared to run unit tests automatically. This has expanded to include the application of continuous QA processes. This improves software quality, reduces time to market and builds a solid foundation for the code’s future.

Who and what is Jenkins?
Without Jenkins – a server based, open source CI tool written in Java – it can take hours to perform an incremental build of an application and tests can take weeks to run. Jenkins enables continuous testing each time a source code change is made.

Introduced in early 2005, Jenkins (see above) has more than 400 plugins that allow it to be used with other coding languages to support the

Author profile:
Niroshan Rajaduari is director, EMEA, with Vector Software.

VectorCAST/Manage project
- PowerPC tests
- Windows tests
- ARM tests
- Linux tests

Physical or virtual machines
- Jenkins nodes
- Jenkins nodes
- Jenkins nodes

Master server
(master)

EMBEDDED DESIGN
EMBEDDED SOFTWARE DEVELOPMENT
build and test of any project. Jenkins is a ‘job server’, with no bias as to which job needs to be performed.

Adding parallelisation
Organising tests is one part of the CI process, but parallel testing is also crucial. When using Jenkins as the CI server, test targets need to be scrutinised and a popular choice is Wind River’s Simics, which can replicate a mode of the target hardware.

Using Jenkins and Simics, engineers can select which environments to test and discover which cases need to be rebuilt and run, based on source code changes. Engineers can set up different configurations of the same board to run comparable tests, allowing for complete code coverage testing.

Simics makes testing more reliable, as having a model of the hardware – sometimes before it is available – means there is less chance of encountering errors with actual tests. Simics can also record and replay input values, which supports CI by allowing the information to be accessed and used by another team member.

A complete CI environment needs a platform to bring these elements together. The ideal solution will organise all test cases into groups that allow developers to map the application’s architecture and for individual stacks to be tested and pushed forward for system tests.

Test what needs to be tested
With an environment that can carry out a complete suite of tests as quickly as possible, further improvement can be obtained by carrying out only those tests required to restore 100% completeness when a change is made to the source code. The well known principle of change impact analysis is the final ingredient.

Example project
VectorCAST has an example project of 20 environments with 122 unit tests. The test environment (see fig 2) contains VectorCAST, a Jenkins server and nine slave nodes distributed over three Simics test hosts, each with three simulated board environments. We know the baseline build and execute time is 47mins using one slave node and a Simics board to run a complete set of tests.

To demonstrate the power of parallelism, 20 environment test jobs are created using VectorCAST, which pushes them into the Jenkins build queue. Jenkins pushes the first nine jobs to the slave nodes, where they execute on the Simics boards. This continues until the full build is complete. The result is a full distributed rebuild time of 7min 47s – six times faster.

If we change two modules and start a rebuild, VectorCAST and Jenkins determine the tests that need to be rerun, then push the jobs into the queue. The first module needs a complete rebuild and a test of the three test cases; the second module needs only an incremental rebuild, executing two of the nine cases. This requires only five of the 122 test cases to be rerun and an execution time for the rebuild of less than 2mins.