LDRA
Software Technology

Unit Testing Embedded Systems on Railways

Article by Bill StClair originally published in Automotive Electronics Magazine June/July 2005
automotive electronics

Unit Testing Embedded Systems on Railways

Bill StClair of LDRA

The rail industry can learn from the aerospace business in using software development tools, says Bill StClair.

Since the Industrial Revolution railway technology has been an engine of growth in the European economy. Plodding coal-fired engines have been supplanted by rail systems that ride electro-magnetic waves and make safety-critical decisions in milliseconds. In order to keep pace with rising demands of the railway industry, both technologically and financially, the software that drives today's railway systems must be thoroughly tested by the producers of such systems and not by the customers and citizens that depend upon them.

Fortunately, software engineering methodologies have been evolving to meet these demanding criteria and as new techniques have become available the rail industry has been amongst the first to take advantage of the benefits they bring. A strategy for testing software is now seen as critical as software design and implementation. Certain development methodologies, such as Extreme Programming, require test designs be specified before implementation begins. Notwithstanding the development methodology you now deploy, today's software testing requires an enforceable strategy that can be applied consistently and repeatedly across your entire software system.

A strategy for testing software is now seen as critical as software design and implementation.

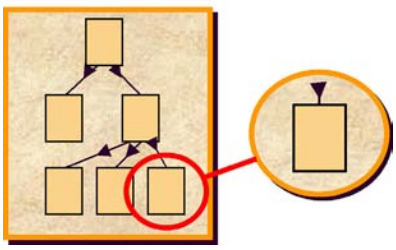
A proven strategic approach to testing which has become commonplace within the railway industry includes a thorough yet complementary combination of Functional and Unit Testing. Functional Testing includes the demonstrated capability of your software to meet customer requirements. This category of testing is typically performed at the system and/or subsystem levels; it is highly procedural, consisting of hundreds of "steps" and is part of a top-down process of system validation.

However, Functional Testing alone is rarely sufficient. This insufficiency is due in part to the obvious precondition that the system (or subsystem) under test must be coded and functional before testing can begin. If you consider the merits of iterative development and its ability to focus on the parts (or components) of your system in order to effect a modular design, another testing technique must be employed. This technique, called Unit Testing, is a bottom-up process that focuses on system internals, such as classes and individual functions. Not only does Unit Testing facilitate early stage or prototype development, it can also be used to cover the paths and branches in your code that may be unpredictable or otherwise are impractical to exercise from a Functional Test perspective.

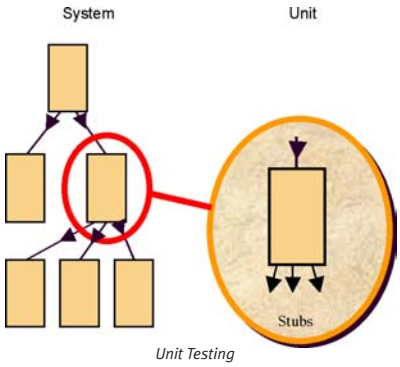
Defining Unit Testing

It is not always clear exactly what is meant by the term Unit Testing. People often talk about Unit Testing when in fact they are actually referring to the similar process of Module Testing, as illustrated in the following diagrams.

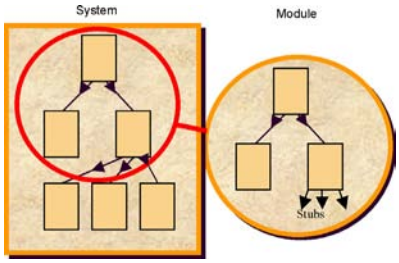
Put simply, Unit Testing (Figures 1 and 2) is a process of testing an individual software unit (function, method, etc) in isolation while Module Testing (Figure 3) allows for the simultaneous analysis of several units, i.e. a multi-unit test.



Unit Testing
Figure 1



Unit Testing
Figure 2



Module Testing
Figure 3

There are two primary activities in a Unit (or Module) test: Interface testing and test verification. Many projects focus the Unit Testing process on deriving inputs and expected outputs from a requirements and design base and then utilising this information to determine if the set of actual outputs matches those predicted.

"The problems associated with traditional, manual methods of Unit Testing are numerous and studies have suggested that the technique is under-utilised by up to 90% of software developers as a result."

In addition to the interface technique of Unit Testing, projects often apply different analysis levels. For example many projects apply Unit Testing techniques simply to achieve coverage analysis and have no interest in the actual input or output values associated with this analysis. Typical coverage requirements may include Statement, Branch, MC/DC and LCSAJ path coverage.

Other additional analysis levels that may or may not be applied include various types of stub analysis such as the monitoring and validation of input parameters passed to stubs and the collating of statistics relating to the number of times each stub is called.

Real Time and Embedded Systems Testing Challenges

The problems associated with traditional, manual methods of Unit Testing are numerous and studies have suggested that the technique is under-utilised by up to 90% of software developers as a result.

The following describes some of the key problem areas:

- Lack of a unified and structured method means that techniques are applied on a project-by-project basis with little opportunity for the development of industry-wide standards.
- Increased overhead associated with maintaining and changing manually generated driver programs and scripts in response to changes in application code.
- The complex nature of many test harnesses requires highly skilled engineers and also means that the harness code itself may contain programming errors that have to be debugged be fore the code unit under test can be analysed comprehensively.
- Manually checked regression processes are susceptible to human error.
- Changes to the application code may introduce changing requirements and necessitate modification of the associated test scripts and drivers.

Coupled with the generalised issues listed above, the Unit Testing of real time and embedded software applications raises a series of more specific issues. These include available memory, timing considerations, communication links and software/hardware considerations. Working within specific target memory constraints is not something that is particular to Unit Testing. However it is the norm that processor resources are used to the maximum or near maximum by the application code and this does give rise to additional issues that do not occur with the Unit Testing of non-embedded systems.

For example, greater consideration must be given to the 'physical' size of the driver programs and associated test cases. These must be kept to a minimum to ensure that they will actually fit into the target environment.

This is particularly difficult given the greater degree of rigour and hence test input/output that is required for the analysis of safety-critical systems. While driver code that facilitates Statement, Branch and MC/DC coverage can be easily compressed, this is not the case when there is a requirement for test path (LCSAJ) measurement. Add to this the requirement of some software standards to provide a repeatable process capable of demonstrating compliance at both the source and object code levels, and it is clear that target constraints can have very serious implications for the overall test process.

Automated Unit Testing

Many of the problems associated with the implementation of traditional, manual Unit Testing processes are concerned with the high skill levels required and the considerable, additional overhead that such techniques can impose.

Automation of these processes with the use of tools enables the techniques to be made more standardised yet intuitive. These are highly desirable goals with potential benefits of increased efficiency and reduced costs.

Automation of the Unit Testing process permits the development of repeatable processes and the standardisation of testing practices. Often tools provide facilities to enable the capture and storage of complete test information that can be held in a configuration management system with the corresponding application source code and retrieved and imported at a later date for regression testing. These facilities support outsourcing and audit with the opportunity to introduce greater efficiency and reduce costs in the overall software development process.

Automated tools facilitate greater degrees of control and management of the Unit Testing process as a whole. This can include, for example, the management of test data with the storing of default values in data dictionaries and other techniques to ensure information is available to project team members in a uniform manner.

The use of automated tools greatly enhances the testing of embedded systems. Such tools implement facilities to assist the user to reconfigure I/O channels and to enable testing of software applications in a variety of host and target environments. Tool facilities may be further extended to assist with the management of information retrieval from the chosen execution environment. This can help to remove some of the more difficult configuration issues associated with the Unit Testing of embedded systems.

The experiences of the railway industry demonstrate that Unit Testing as a technique has a great deal to offer the developers of embedded software applications that are seeking to test these systems to the highest possible standards. Add to this mix the growing number of tools that provide the ability to apply Unit Testing techniques with high levels of automation and you have the potential to achieve significant improvements in the safety, quality and integrity of such systems to the obvious benefit of developers and, more importantly, the people that depend on them.

Bill StClair is a Technical Evangelist at LDRA

LDRA Headquarters
Portside, Monks Ferry,
Wirral, CH41 5LH
Tel: +44 (0)151 649 9300
e-mail: info@ldra.com

LDRA Technology Inc. (US)
Lake Amir Office Park
1250 Bayhill Drive Suite # 360
San Bruno CA 94066
Tel: (650) 583 8880

www.ldra.com