HARMAN/BECKER AUTOMOTIVE SYSTEMS

HARMAN/BECKER AUTOMOTIVE SYSTEMS GMBH · BECKER-GÖRING-STRASSE 16 · 76307 KARLSBAD · GERMANY

Test-Driven Development for the Embedded Systems Space

Author: G. Harris

Date: September 14, 2006

Position Statement

All too often in the software industry one sees efforts that focus only on quality considerations that are visible to the end user while neglecting internal quality. Even worse, one often sees efforts to add quality at the end of the development cycle. Software quality is a goal that cannot be achieved unless it is continuously pursued throughout the entire development and maintenance cycle.

Test Driven Development and Software Quality

How does one achieve software quality? The first step is to provide a clear definition and ensure that this definition is objectively measurable. Unit testing is an essential element in such a quality strategy. Unit tests are the sensors that allow a clearly defined quality standard to be objectively tested. Such requirements as ‘at least 50% code coverage’, ‘90% functional coverage’ or limits on resource consumption of any kind, can all be measured objectively and reliably with unit tests.

A more subtle benefit of test-driven development derives from the fact that it reduces oscillations in the development process. Any functionality or behavior pattern that is anchored by a unit test cannot be unintentionally lost or damaged in future development. Such test-supported software provides a firm and reliable foundation for development and is of inestimable benefit in the rapid development processes that are the modern norm.

Test driven development has social aspects, management aspects and technical aspects. An effective use of test driven development will require guidelines that address all of these issues. One possible set of guidelines could be:

- Initial development time must be extended by at least a factor 2 to accommodate test development. The savings will come downstream, the costs are up front.
- A test run must be easy to start. A success must be quick and easy to evaluate. A point of failure must be quick and easy to find.
- Every developer assumes the responsibility to be able to prove objectively that his or her software product does what it should with an acceptable cost in terms of computing resources. The mechanism of choice for providing this proof is test software.
- No function, interface or behavior may be introduced without associated test software. The test software is a project deliverable.
- All developers assume common ownership of all test software. No development effort is completed until all test software functions fully and correctly on all target systems.
- When developing test software, it is essential to include tests of boundary conditions, fault handling and behavior patterns. This is what is often neglected in development and is where vulnerabilities have their roots.
- When correcting errors, always first extend the test software to demonstrate the fault. When the test no longer fails, the fault is corrected.

It is clear that these guidelines describe an ideal that is not always fully achievable. Pragmatic interpretations will be necessary when these basic rules can’t be made to fit. Nevertheless, it is important to publish these ideal principles as goals.

The underlying justification for test-driven development as a general principal is clear. It gives us the ability to provide quality guarantees for software and is an essential step in the transformation of software development from an artistic endeavor into an engineering discipline.

Test-Driven Development in the Embedded Space

What is special about the embedded space? Why is it justified to treat test driven development differently in the embedded space? The key differences can be quickly summarised:

- Embedded software is running in a resource limited environment (weak CPU, limited memory, limited or unreliable power supply, limited or slow back-up storage).
- It is usually running in the same address space as the OS kernel (if any).
- In the field, it is subject to random external stimuli that are difficult or impossible to predict or model in the lab.
- It is often subject to real-time constraints that are difficult or impossible to meet with modern PC's.

Due to these constraints, the behaviour of an embedded system is rarely describable as the sum of the behaviours of its parts. Resource competition, unexpected stimuli and real-time interactions can and will result in faults and failures. Furthermore, it is impossible to observe the system without changing its behaviour.

An individual test could consist of a test program, a sequence of input stimuli to be simulated, and a filter description that is used to evaluate the validity of the output. The test program must be easily downloaded to a target system and started. If provided, the external stimuli must be inserted from the host system (user input or other). Output must be retrieved, cross referenced to the inputs, stored and filtered to generate a report. Success reports should be as trivial as possible, failure reports should be as detailed as necessary (but no more) and should provide references to the raw output to make further analysis easily possible.

It must be trivially possible to assemble a sequence of tests to form a test suite. It would be valuable to add control structures to a test suite to support extended testing (e.g. do these additional tests if a failure was seen), repetitions, loops and the assignment of individual tests to specific target systems.

An environment to support test driven development for embedded systems must support automation of complete test suites.

In order to make the migration of existing test software into an integrated environment as simple as possible, the coding constraints on the test program itself must be kept to a minimum. Ideally it must be possible to take an arbitrary existing test program and integrate it into the new test environment.