Automated testing of digital television devices

A Cabot Communications white paper

This paper discusses the benefits of introducing an automated test framework for digital television receiving devices. It then focuses on three key areas for which automated testing can provide benefits that are difficult to achieve by manual testing alone: conformance testing, engineering support and resolving corner cases.

The challenge of testing

With ever increasing numbers of product types with multiple features in the digital television industry, testing is rapidly becoming more challenging. Manufacturers provide television products for terrestrial, cable, satellite, internet broadcasts/streams, catch-up TV, internet/home connectivity. Services can be free to air broadcasts, subscription services, on demand services, encrypted services. Hardware can include set-top boxes, integrated devices, recorders, media players, CAM devices, USB devices. Hardware and software features need to work individually but they must integrate with each other.

In addition, each individual geographical market requires devices to conform to its particular standards. For example, in the UK, it is highly desirable for a manufacturer that its digital television products pass the conformance tests of the Digital Television Group in order to gain certification of compliance to the D-Book, which is the technical specification for digital terrestrial television in the UK. Such products are given the digital tick and Freeview logo.

This increasing technological complexity can make automated testing a daunting prospect. Perhaps it is an easier prospect to continue to manually test rather than to convert your manual test scripts to automated scripts. A digital television, for example, can require 20,000 to 30,000 tests, and writing this number of automated test scripts is a considerable effort. However, as this paper will show, after the initial investment into automated testing – installation, set-up and script writing/conversion – significant savings of time and effort can be realised. The key reason for this is that automated testing saves time for certain types of test. But there are many other benefits of automated testing.

Automated testing of digital TV products

An automated test framework (or test harness) is a system that sends commands to the device under test and observes its behaviour and outputs. For automated testing of digital TV products, software executes test scripts via a software-hardware interface. A comparison then takes place to determine whether actual results meet expected results.

It performs laborious tests automatically, quickly and reliably. However, it is not a replacement for manual testing: it augments manual testing, allowing testers to focus on investigation, analysis and increased coverage.

Upon introduction of automated testing a significant amount of time is required that would normally have been used for manual testing, which instead is spent installing and configuring the framework, training staff and writing tests. So automated testing does not immediately reduce the test schedule. However, by the end of the first few test cycles, it is likely that you will be level on test effort compared with manual testing, and from then on running further iterations of your automated tests will offer significant time savings.

The best types of test to automate are:

Conformance tests – by avoiding human error, automated conformance tests are repeatable and reliable.

Regression tests/repetitive/labour intensive tests – execute large numbers of test cases over and over again, without operator fatigue.

Stress tests – test stability by sending a command to the device repeatedly, perhaps thousands of times.

Soak/endurance tests – test stability with a heavy load over a sustained period of time. Again, difficult or tedious to do manually.

Time saving

It takes typically 5 to 15 minutes to carry out a test manually, and around 30 minutes to write and validate an automated test script that will perform the same testing effort. So although testing manually is quicker initially, the more test runs performed the more efficient automated testing becomes. If you are running tests on a nightly basis (against a nightly build for example), automated testing rapidly becomes highly cost-effective.

The table below shows some real-world examples of times taken to complete test runs on a digital television, both manually and using an automated test system.

Task	No. of tests	Time, manual testing	Time, automated testing
Regression tests	3,000	15 working days	6 hours, overnight
Full test	20,000	100 working days	40 hours, continuous 24/7 testing

In addition, a good automated test framework will store the results of automated tests automatically and will provide a report generation feature to output those results in a range of formats. So test results are available through minimal time and effort.

Other benefits of automated testing

In addition to time saving, automation provides several other advantages over manual testing:

- Consecutive test runs are exact duplicates of previous ones.
- No human error (once test scripts have been validated).
- 24/7 testing.

- Reliable interpretation of results.
- Testing under conditions that are otherwise difficult to simulate (stress testing, for example).
- Free up time for investigation and analysis.
- Repetitive tasks are done by machines.
- Tests written by skilled engineers can be run any number of times by anyone operating the system, regardless of their level of skill.
- Cost effective for devices that require continual and large numbers of test runs, with the potential for significant cost savings in the long term.
- Can test a large number of devices at the same time.

Downsides of automated testing

The disadvantages of automation relate to the increased costs associated with the following:

- initial investment
- time to install and configure
- time for training on the system
- time to write automated tests
- maintenance of tests for new or changed development features

However, these can be quickly recovered over the course of multiple test runs. A good automated test framework capable of providing engineering support and reliable conformance testing will pay for itself within 12 months. These negatives can be overcome with an effective implementation plan and high quality support from the chosen supplier.

How automated testing helps conformance testing

When attempting to perform conformance testing manually it is difficult or even impossible to be fully confident that you can repeat tests precisely. Discrepancies inevitably arise such as timing differences, set-up differences, or tester error.

Every manual tester will have their own view of what constitutes a pass and what constitutes a fail for a particular conformance test. So the interpretation of results from manual testing differs from engineer to engineer and may differ further from the interpretation of conformance test results by a formal test house.

Conformance tests are ideal test types for being automated, by allowing test engineers to quickly and – more importantly – reliably determine whether or not the product is compliant with the relevant standard, and to see where things break so they can be fixed. By avoiding human error, automated conformance testing is consistent and reliable, reducing the time taken to achieve compliance. With automated tests interpretation of results is reliable and consistent from one test run to the next.

Once conformance tests are automated, test engineers can devote more of their time to investigation, analysis, increasing test coverage and resolving corner cases.

Writing automated conformance tests

Test suite documentation can be obtained from the conformance standards body. The test suites describe everything that the device under test must do to comply, and these form the test project requirements.

Typically the types of errors in a product encountered during conformance testing are timing errors or screen capture errors, so you write tests that determine whether an event has occurred at the correct time, and that the correct information is displayed. When analysing screen output it is often preferable to examine a portion of the screen rather than the entire screen, because some parts of the screen may have moving video or other information that may change independently of the portion of interest (a banner, for example). You can examine portions of the screen by using mask images to mask out the part of the screen that is not of interest. The test should determine whether or not the information examined is in the right position and of the right size, usually to a tolerance of one pixel. For text, optical character recognition is used to verify if a string is correct.

There are three image capture/comparison algorithm types that can enhance your ability to test screen output effectively:

- an image capture/comparison algorithm with **tolerance** to slight offsets or image quality;
- an image capture/comparison algorithm that can capture foreground images in a **transparent** frame, which is typically overlaid on a background of moving video (such as a menu display);
- an image capture/comparison algorithm that can test the accuracy of the software's ability to plot **cross hairs** centred on targets.

These types of algorithms are discussed in more detail below.

Position offset/image quality tolerance algorithm

When testing hardware with a stream, you need to be able to take into account a slight degradation of the quality of the images due to inherent losses in systems (especially those due to lower quality connections such as SCART).

This type of algorithm uses image blurring to allow captured images that are of lower quality than the reference image to pass in comparison tests. If a standard comparison algorithm were used the test might report a failure. Similarly, you can apply a tolerance when comparing the position of the capture so that an image that is slightly offset compared to the reference image passes. For verifying position, if the captured image is not matched first time, the captured image is moved around within the specified offset tolerance to attempt to determine a match. The acceptable offset tolerance may be set by the conformance standards body and then applied to the automated test.

Analysing transparent frames with moving video background

This type of algorithm is used to test the foreground layer of an on-screen display when the background is moving video. This is typically the user interface which is in a frame which has a degree of transparency. The algorithm uses colour analysis to separate the frame colour from background colours.

Plotting cross hairs

This is an algorithm type that tests the ability of an MHEG engine to plot bitmaps on screen at precise locations. A tolerance is usually permitted, defined by the standards body, and is typically set to one pixel. A conformance standards authority will issue test streams in which targets are drawn at various locations on a background of moving video. The MHEG engine must plot a crosshair centred on those targets. If the crosshair is one pixel offset the test is passed, but any more than that and the test fails.

Validating conformance tests

So you write a test, you run it and the test result is a pass. How can you trust this result? How do you know that your tests themselves are valid and that they correctly determine whether or not the device meets the conformance standard?

The solution is to test your test by stressing the device under test to produce errors and then verifying those errors. Send commands that produce failures and commands that produce passes so that you determine that the test can be passed with suitable commands/inputs.

Corner cases

Many bugs are easily reproduced and their cause is known immediately or with relatively little investigation because a simple function or process has failed. However, when a number of edge cases combine at the same time and produce an error – known as a corner case – it can be difficult to reproduce this due to the number of different inputs and processes involved. Corner cases are produced when different functions of the system interact and produce unexpected behaviour.

A common situation in which to find corner cases is after a cold boot of a set-top box, when various states can occur which might show up problems in less stable software. For example, the review buffer failed to start, the set-top box performs a disk check which causes some timing delays, the set-top box takes too long to start up completely, a network change occurs, an over-air download takes place. Because the state of the device can vary like this just on start-up, intended normal working functionality of the device can be prevented if the software does not handle these states. Time is needed to manually test the functions of the device when the device is in these various states following start-up and to investigate any errors encountered. Such tests are not only successful as part of product testing, but can also be incorporated into a test-driven development environment in the early stages of development.

You cannot write test scripts to find corner cases: you would normally suspect a corner case when unexpected behaviour is encountered that is difficult to reproduce (because there are several conditions that need to be in place in order to do so). Information comes from the engineering department that there's a problem in some part of the product. Sometimes a software engineer has written their own code test (unit test) which has thrown up an issue that requires other forms of testing, such as integration testing. Or a test engineer simply notices a problem in one area of the product under test during the course of testing a different, but interrelated, area. The test engineer and the software engineer will often collaborate to develop a suitable test.

Automated testing can help to resolve corner cases by allowing normal testing to continue using automated test scripts, while test engineers investigate and solve the corner cases by use of manual testing.

How automation provides engineering support

In digital television, engineers are required to switch projects often because time to market is critical and product features compete with one another in the race to market. Getting one product with one feature set to market can mean another project is put on hold or dropped. Constantly changing priorities means long-term planning is often not possible, so any process that can respond quickly to new developments or opportunities is likely to be successful. Automated testing can provide key support to an agile engineering development environment by giving near instant feedback on new development, allowing bugs to be found on a continuous basis, quicker and to be fixed more easily because the work is fresh in the mind of the engineer. In contrast, with a traditional test approach, which might only do a full test monthly, it is more difficult for engineers to remember the intricacies of the work they were doing in that area. They may have moved to other projects, or, worse, left the organisation. A month delay might require the engineer to spend five times longer than usual to remember what they were doing.

If your software engineers adopt a continuous build process in tandem with continuous smoke test and overnight automated unit testing, they can be sent an email or SMS immediately when defects are found so bugs can be fixed much more cheaply.

As an engineering tool, automated testing can offer 100 per cent reproducible testing, especially useful for conformance regression testing.

Automated testing is also suited to white-box testing but you need to know a lot about your system. You need comprehensive documentation of your code to know how to test its internal workings. Black-box testing is usually sufficient for conformance testing, which requires that the functions of the device and what it outputs conform, rather than how it does it.

A typical process for nightly build regression and conformance testing:

1 Code changes are made during the normal working day.

2 The new code is built at the end of the day (nightly build).

3 Automated tests are run against the nightly build. Tests can be prioritised to run the most important tests first (perhaps the tests relating to an area of code that has recently been reworked).

4 Dashboards are automatically created and are ready to be examined in the morning. These list each test suite, each test within it, and whether it passed or failed.

Nightly automated conformance testing can be done in two hours, versus two and a half days for manual testing.

Ten things that make a good digital TV automated testing framework

1 It should be a specialist product, specifically designed to test digital TV products, rather than a generic automated testing framework.

2 Ability to test multiple devices simultaneously. This allows you to be creative: run the same tests on different types of device (such as different television models); run different tests on multiple devices of the same type at the same time to reduce the time required to carry out a given test run.

3 Central database and test servers to store all your requirements, test plans, tests, scripts and results centrally; update in one place.

4 Sophisticated image capture and comparison algorithms.

5 Project requirements management – it is useful to be able to link test project requirements to tests and results ensuring easy visibility of test coverage. This allows project managers and test engineers to collaborate effectively to ensure that tests have been written for each requirement and to verify that the tests are sufficient to meet the requirements.

6 Workflow process built-in, to manage user responsibilities, authorisation and approval. This allows testing effort to be monitored and ensure that things are done by the right people, in the right order, at the right time. It can be used to prevent unauthorised access to certain parts of the software and unauthorised editing of data.

7 Ease of installation and equipment configuration.

8 Client software that can be run remotely over an IP network. Allows the test engineer to be physically separate from the automated test equipment and the device under test, if required.

9 Connection to bug tracking systems, email servers and/or SMS system. When a test fails, a defect report can be automatically created and assigned to an engineer or group of engineers and email/SMS notification can be issued.

10 System reliability backed up by an effective support, maintenance and development road map.

Automated script writing

This section discusses some good working practices and approaches to writing test scripts.

Best practice

Often a manufacturer's range of television devices has many features in common across the range, such as the user interface design. Product variations come in the form of recording ability/method, internet connectivity, media browsers, and many other feature sets. Digital television products can overlap in their feature sets, which means that test suites can also overlap. Scripts can therefore be reused in multiple instances and for multiple products.

Tests should be written in a modular fashion so that groups of tests can be run when required but each test can be run also in isolation and verified independently. When an individual test script can be run by itself with no dependencies, maintenance is also greatly simplified. This can be achieved by writing test scripts that always return the device to the same point at the end of each test.

A test script should cover a single known requirement of a single defined function.

Maintenance of test scripts

Updates are desirable as part of routine maintenance and improvements, but are mandatory when a standards body issues an update of its conformance test suites. Ease of test script maintenance is therefore highly desirable. Separating out key presses from test scripts is an excellent way to ease maintenance.

Macro mechanism for key presses

In this technique, common commands like key presses relating to the navigation of a particular user interface menu structure are stored in a separate 'macro' file, which can be reused by any script that needs to perform key presses on a device that uses that menu structure. The script uses macros to call commands or key presses from the external macro file. In this way, you can make changes to scripts independently of the key presses that are specific to different user interface menu structures.

When a device's user interface menu structure changes, you only need to modify the key press macro file, not the entire test script.

It also makes it easier to update the framework when you need to test devices that use different user interface designs: if a different device is used, you only need to point to a different key press macro file.

Synchronising scripts and test documentation

It is important to keep your test scripts synchronised with test case documentation, which is the description of the test procedure to follow, because if a script needs to be debugged or changed, it is helpful to be able to refer to the documentation to know what the script is designed to do.

If test case documentation is accessible from the same place in the automated test framework user interface as test scripts, it is easier to keep these two objects synchronised (for example, click on a tab within the test case definition to access its test script).

Test scripts and test case documentation are two different things, which can really only be mutually consistent by being diligent and having a good process in place that tracks changes and has some form of managed approval. When changes are ready for approval, the approver can verify that the script and the documentation are coherent and that they both reflect the current developmental state of the device under test.

Cabot's Robotester system

Robotester is a specialised automated test framework designed to test digital television devices. It provides the following features (not an exhaustive list):

- Advanced patented screen capture and comparison algorithms and methods.
- Centralised database for test scripts.
- 'RoboHub' easy capture device configuration – all in one box, IP connected. See below.
- Fully scalable and extensible.
- Client/server model to allow testers and devices to be in separate locations – client requires only a network connection.
- Networked stream player that can play recorded streams.

In addition, Cabot provides:

- Off-the-shelf conformance and functional tests.
- Services: for example, full commissioning of a Robotester system, support and maintenance, platform customisation and new test development, product testing and pre-certification services.
- Expertise, knowledge and experience of working with national and international broadcast, testing and conformance standards.

RoboHub data capture device

In order to capture the different types of data you encounter with different device types – HDMI, SCART, LVDS, audio – you need different capture devices. RoboHub integrates three types of capture card into a single device. It is more cost-effective than buying three separate cards and you don't need to install three separate cards into a PC. It makes connecting devices much quicker. Each transport type has a different port. Running tests is easier because you refer to a single device (RoboHub) rather than three different capture cards. RoboHub is controlled via the IP network.

Contact details

For more information about Robotester contact Cabot at:

www. c a b o t . c o . u k

Cabot Communications Ltd, Verona House, Filwood Road, Bristol BS16 3RY, UK

Tel: +44 (0)117 958 4232. Fax: +44 (0)117 958 4168. Email: info@cabot.co.uk