How Smart Are Your Products?
Testing Methodologies for Information Appliances

By Michael Goodman, Andy Goryachev

Gone are the days when Information Appliances were designed to perform a single task. Looking back, it was so easy to design a product to do one thing and do it well. A pocket AM radio is a good example. The feature list is short and the behavior is predictable. Granted, there are numerous ways to build an AM radio, but the end result is mostly the same. It converts AM to audio, and we can easily verify how well it does its job. Testing was simple.

Designers of Information Appliances are faced with a much bigger challenge today. An Information Appliance is first and foremost a computer, which, in most cases, runs an embedded OS. Its programs can take it in a variety of directions, some of which may be entirely unpredictable. Let’s say you are designing an Internet Radio. Most consumers will expect that it will also rip and burn CDs. Of course, since it is a network appliance, it will have to speak TCP/IP and will probably feature a USB host port, which carries along the corresponding drivers and protocol stacks.

To improve the customer experience, you will design a user interface and program a series of menus for the LCD screen. You will define a map of states and behaviors, depending on the exact functionality of the device. You will create and manage a number of streaming and compression algorithms. You’ll try to squeeze in network upgradability, so the device could handle the brand new, still-in-draft-version communications protocol when that comes out. Add a couple of last-minute features and the complexity of the design becomes substantial.

Now imagine that you have to test this device at the trailing edge of an already condensed project schedule. Will everything work as expected? Will you have the time to fix it without missing the deadline? Is you product as smart as advertised?

Although it seems risky to leave little time at the end of the project for testing, a surprising number of manufacturers fail to account for this important process early in the development cycle. The results are, unfortunately, all too familiar: confused customers, increased burden on the service department, projects that don’t seem to end – all leading to mounting support costs after product’s release. Add to that the potential brand damage due to market perception of “vaporware” and you soon realize that the real cost of incorrect testing is often underestimated.

There is also the problem with having tests performed by the original developers. In addition to being emotionally attached to the product, developers understand it too well and will only test the logical sequence of events. It’s unnatural for the developer to imagine new, “untypical” scenarios. There is a risk that some scenarios will never be tested until the product is in the hands of the less-technical customer. To mitigate this risk, you need a comprehensive test plan, preferably written by an independent third party, which is not involved in the design of the product.

We live in an era of software-based devices, which are dangerously complex. Their complexity requires adding a range of new testing methodologies to the standard array of tests to which we subject our products. Modern software requires a testing process that is fully integrated with the development cycle. Below we list the components of a robust test plan for a mission-critical application, such as SEI level 5 software. Depending on the environment in which your product is used, the list may be paired down, but beware – compromises in testing may result in compromised product performance down the line.

Functional Testing
Requirements for the product are translated into an engineering specification, complete with specs and technical parameters. This is what engineers actually build. Functional testing is a process that seeks to find discrepancies between the actual product and its engineering specification. The following items should be validated in addition to the main requirements:

1. Look and Feel:
   - Font types, sizes, colors, display options
   - Color contrast and compatibility
   - Data field consistency (includes data fields, icons, data itself, etc.)

2. Navigation and Keys:
   - Scene navigation / key navigation
   - Valid and Invalid key pad entry
   - Reverse screen sequences

3. Screen Layout / Contents and Sequences
   - Is there a match with the documented requirements and specifications?
   - Are navigation links correct?

Scalability Testing
If the device includes an OS, it most likely has a file system for storing internal and external data. A well-designed file system should be able to handle a sufficiently large number of files to prevent “surprise” crashes, due to internal overload. The OS itself should be able to reliably spawn all necessary threads (programs). Application and driver software needs to respond to a sufficiently large number of stimuli without appreciable system slowdown.

Scalability testing addresses situations where the number of entities the software operates on increases dramatically. Initial assumptions about number of files, storage size, request frequency, etc., may be found incorrect when the product is deployed. It is important to verify that the product can scale without taking a hit on performance.

Performance Testing
Most signal processing and many other applications have specific performance or efficiency requirements, listing such properties as response times and throughput rates under certain workload and configuration conditions.

1. Performance Testing has the following objectives:
   - Demonstrate that the system meets specified performance objectives
   - Tune the system

- Determine the factors in hardware or software that limit system performance
- Project the system's future load-handling capacity in order to schedule its upgrades and replacements

2. Performance Testing assumes a robust, working, debugged, and stable system. Given such a system, Performance Testing has the following components:
   - A clear statement of performance objectives
   - A source of transactions to drive the experiment
   - A controlled experimental process or test bed
   - Instrumentation to gather performance-related data
   - Analytical tools to process and interpret the data

3. The following items should be included in Performance Tests:
   - Initial application boot up time
   - Key navigation time
   - Scene transition and updating time
   - Pop-up window time
   - Server connection time
   - Feature downloading time
   - Bandwidth and latency of application update
   - Bandwidth and latency of downloading the new data modules
   - Bandwidth and latency of internal communications and data transfer

Usability Testing
The goal of Usability Testing is to evaluate human factors on the following criteria:

- Does the UI match the intelligence and environmental pressures of the end user?
- Are the outputs of the application clear, meaningful, and non-abusive?
- Are the error diagnostics (e.g., error messages) straightforward and intuitive?
- Does the total set of user interfaces exhibit "conceptual integrity", an underlying consistency and uniformity of syntax, conventions, brand messaging, semantics, format, style, and abbreviations?
- Where accuracy is vital, is sufficient redundancy present in the input?
- Does the application contain an excessive number of options, or unlikely options?
- How quickly does the application return an acknowledgement for all inputs?

Stress Testing
Stress Testing is a process of applying a high background load, to the point where one or more, or all resources are simultaneously saturated. It is to system testing what destructive testing is to testing of physical objects. The intention of a stress test is to "break" the system, to force a crash.

Stress Testing should accomplish the following objectives:

- Force race conditions
- Totally distort the normal order of processing, especially processing that occurs at different priority levels
- Exercise all system limits, thresholds, or other controls designed to deal with overload situations

- Greatly increase the number of simultaneous actions
- Deplete resource pools in an extraordinary sequence

Error Handling Testing
Error Handling Testing evaluates the following issues:

- Server data download error handling on the application side
- Application on-screen help messages (or pop-up error message windows)
- Data corruption recovery
- Recovery from hardware configuration changes

Integration, Installation Testing
When installing the application, the user must select a variety of options, files and libraries must be allocated and loaded, a valid hardware configuration must be present, and the application must be connected to the server, if a client-server configuration is used. The purpose of the Installation Test is to locate any errors made during the installation process.

Among other things, the test cases might check to ensure that a compatible set of options has been selected, that all parts of the system exist, that all files have been created and have the necessary contents, and that the hardware configuration is appropriate.

- Test the installation procedures and validate installation results
- Test the integration of client application and server application
- Verify lack of system corruption due to incorrect installation

Documentation Testing
A thorough review of user and service documentation is necessary to ensure conformance with actual performance. This includes validating the user manual, examples, and verifying the warning and error messages.

This list is by no means all-inclusive and is offered as an example, to highlight the issues to consider when creating a robust test plan for any software-based product.

Savvy product developers make it their business to test their products rigorously to ensure mission-critical reliability and highest performance. This sharp focus results in recurring predictability of the development effort, allowing your projects to end on time and your engineers to switch to new assignments right away. Overall, this means that the end customer catches fewer bugs, and the cost of service calls is drastically reduced.

If software testing is properly integrated into the development process, you will have the confidence that your development costs are low and your products are as smart as advertised.

###

CEntrance, Inc., a design firm at the forefront of innovation, which develops professional audio and consumer electronics products used in homes, radio stations, recording studios, concert venues, and virtually anywhere else music is made or heard. By paying special attention to product definition, design, and user interface, CEntrance develops products that customers come to rely on-and love. For more information, please contact CEntrance at 847-581-0500 or via email at info@CEntrance.com