Learning to Test Wirelessly

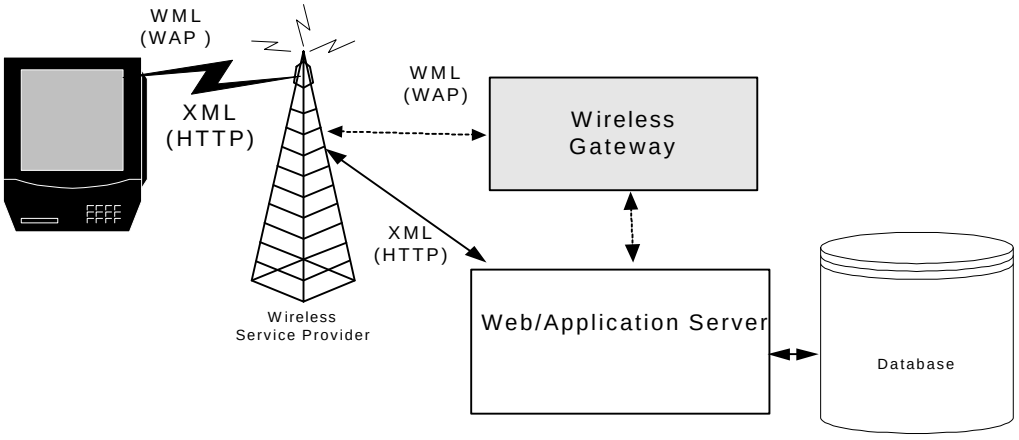
Background

The beginning of my adventure in learning to test wirelessly was accepting the task of building a software test department in a Campbell start-up company. Setting up the test department infrastructure and instigating a review process for design and test documents happened in parallel with analyzing the wireless test requirements and doing the testing. Testing without the infrastructure already in place contributed to some of the process problems that we encountered. For example, the first product was developed with a prototyping development life cycle that did not include design reviews. The test department was expected to operate on a waterfall product life cycle. This disconnect generated a process problem which in turn let design errors through the earlier stages of the product life cycle. Since the design had not been reviewed by anyone (including test), it was not as robust or user friendly as it would have been had it been properly reviewed. **Problem:** The product testing cycle did unearth design problems, but unearthing them at so late a point in the development cycle increased the cost of fixing the problems and lengthened the development time by approximately a month. **Solution**: The solution that we arrived at was to instigate a review process for product design documents and all test documents. This solution helped us to find problems early and foster communications between the departments. **Lesson learned - one**: Including test (and others) up front through design review produces a more robust design that survives the test phase better and cuts the time and cost of development.

This solution of reviewing the design spawned the second process problem. **Problem:** What would be an efficient way to document the design so that we could review it efficiently? Also, with multiple design teams working in parallel, what would make knowledge transfer between design groups proceed more smoothly? While there is background documentation that needs to be done for the developers to code confidently, the graphical user interface (GUI) is what the test department needed documented for their black box testing. GUI documentation also gave the engineers a framework from which to discuss the product functions. **Solution:** We evolved a GUI design document that provided all the typical information needed to do black box testing. Since the template "evolved", changes to the template had to be retrofitted into the other design documents being developed in parallel. **Lesson learned – two:** Having a carefully thought out template defined before teams split out to design in parallel saves time and re-work. (Appendix A: Sample GUI Design page).

Before further discussion, it would be sensible to define what was and what was not tested by the test department. The types of testing that were done elsewhere or were risked included: stress testing, performance testing, security testing, specific protocol testing, and direct Extensible Markup Language (XML) tag testing. So, excluding all that, what was tested? The testing that we did involved: black box function and system testing, usability testing and device/browser combination certification. The certification of a device/browser combination involved the verification that the style sheet (XSL) for specific device/browser pairs worked properly with the formatted data (XML). This testing was performed using the actual hand held devices and browsers rather then emulator versions of those items. Our wireless products are composed of some locally written code and some vended code. The baseline for defining correctness of behavior is the behavior of the non-wireless versions of the products.

Wireless Environment



A simplified version of our product's n-tier architecture is displayed in the diagram above. If the user's wireless device has a browser using Wireless Application Protocol (WAP), then the signal must pass through the wireless gateway which translates the WAP to Hypertext Transfer Protocol (HTTP) for the web server which only understands HTTP. However, if the user's wireless device has a browser using HTTP, then the signal goes directly to the web server. The devices could be any wireless device. For example, it could be a Palm Vx with a modem, a Nokia phone, or it might be a Compaq Pocket PC with a modem.

Introduction

Initially, not having previously tested a wireless product, the test cases were built from Web testing experience. This was a good first draft at the test needs since there were many similarities and some differences between testing Web products and testing this specific set of wireless products. Of those similarities and differences, only those attributes that caused problems will be addressed here. These attributes whose problems we had to solve were:

- Product code is not installed on hand held device.
- Relative immaturity of the Wireless arena protocols and products.
- Screen real estate is extremely limited

Problems encountered and our solutions for them

Product code is not installed on hand held device.

As with typical web based products, our wireless users do not have any of our product's code on their machines. They have a browser which interprets the data stream presented to it by our product. Essentially, our product's code has no direct control of the user's device.

Having no product code installed on the hand held device raised the following issues:

- It left the product totally dependent upon the browser functionality to present the data. Thus each browser displayed a different look and feel due to the different style sheet generated to exploit the browser features. This also led to an explosion in the number of style sheets that needed to be tracked, tested, and maintained individually.

- It meant that for any button/link to function, for message or dialog box to display, or for data to display, a request had to be generated by the style sheet and travel from the hand held device to the application server and back to the hand held device. This round trip to the server for every function impacted usability severely.

- It meant that user access to cached memory was out of our product's code's control and product access to cached memory was trickier to control.

The initial design of the screens took advantage of all typical Web browser functions. However when faced with actually adapting the product to the myriad of WAP browsers with different actual supported functions, the data display changed dramatically from one browser to the next. Each WAP browser supported a slightly different sub-set of the typical Web browser features. For example, on one browser a field might be a drop down selection, but on another browser that didn't support drop down boxes that display became a radio button list, or worse, a data entry field. Eventually, the design was reduced to use a set of functions found in the majority of browsers to be used. Then the majority of the display was the same on all the browsers. However, we still had one style sheet per browser which meant that every change to the design had to be propagated through the system and completely re-tested to insure that all style sheets had indeed been updated correctly.

This use of separate style sheets for each device/browser combination caused an explosion of style sheets. **Problem:** This explosion of style sheets in turn created problems in two areas: keeping track of which version of the style sheets was in the system became a nightmare; re-testing each browser/device combination completely on all the test cases was time consuming, a resource hog and tedious.

Solution: The problems were solved by rolling all the style sheets for a particular view into one style sheet with logic to handle specific device/browser differences. This was possible because of the previous re-design of the GUI into a common sub-set of browser functions. Now, since the bulk of the style sheet was fully tested after the first round of testing, testing for new device/browser combinations involved only testing the new code added to the style sheet to accommodate the new device/browser combination and minor regression testing for the rest of the style sheet. Two added benefits resulting from this solution were: that the style sheets could be analyzed at a later time to give a browser feature comparison of the browsers serviced by the sheets; design updates became much simpler to implement. **Lesson learned – three:** Start development and testing with long term plans for handing the product's complexity, versioning and eventual maintenance.

With one style sheet per view, it was much easier to make changes to the style sheet's design. This was extremely helpful because concern over round trips to the server meant that, for usability, any path to data or a function completion needed to be three clicks or less on average. Theoretically that is an easy, black and white thing to test, but in reality, determining how to average in scrolling (some scrolls required a round trip to the application server and others did not require it), and where to start counting the clicks for a function was not that easy. Putting aside scrolling, since it was easy to know when the scroll involved a trip to the application server, determining where a function path starts is not a clear metric. A function path really starts where ever the user decides that they want to accomplish a function. This means that the path to a function is dependent upon where the user is in the product when they decide they want to perform that function. **Problem:** How do you make the product flexible enough to allow the user to get to data in three clicks from anywhere in the product, not just on predestined paths? **Solution**: The solution to this problem is to make sure that the product has a very flexible navigation system. In the process of testing, even though the product already had a navigation structure of shortcut links, more shortcut links were added to allow the user to change their path choices at many more points. **Lesson learned – four**: Testing for usability means looking past the design both for the wireless product and the desktop product. Testers needed to lobby for adding short cut links at critical places so that the user could get to the function directly even though on the desktop product the user would have had to traverse several screens to perform that function.

The last problem that resulted from not having any code on the device, controlling access to cache, was a much bigger challenge. Each device typically had a way to set cache to zero, so theoretically cache should not be available to become a problem. Because the product required the user to set cache to zero, testing for cache problems was not in the original plan. **Problem**: Many wireless browsers are still able to access a memory cache even when device cache is set to zero. The browsers typically allowed their back arrow to navigate to old data from browser cache. **Solution**: There were two parts to this solution. The browser back button usage by the user had to be "fixed" by having more defensive code in the application server. Then, if the user tried to perform an invalid function on old data like closing a case that had already been closed,. it was handled properly by the move defensive code. The second part of the solution, forcing the browser to always go to the application server for data required more imagination on our developers part. Our lead developer, came up with the idea of including the date and time of the request in the URL. Then, since the browser will not pull the requested page from cache unless the URL of the pages match exactly, there was never an exact match and the request always went to the application server. **Lesson learned- five:** As always, do not forget the negative condition testing. Cache testing needs to be done even when the design seems to eliminate the need for it. Minimally, the testing needs to verify that cache cannot be misused in some way from the product or the browser.

Relative immaturity of the Wireless arena protocols and products.

Wireless being a relatively immature arena caused the following problems:

- There was no clear set of winners for device/browser and device/operating system.
- WAP/WML currently has more limitations than HTTP/XML.

Since the wireless product arena is much less mature then the Web product arena, there are many choices of browsers which each support a slightly different set of features and there are no clear winners yet. The problem that this caused in a prototyping environment was that every time the browser/device changed, the test cycle had to start again from scratch because there was a whole new set of style sheets. As mentioned earlier, we had an explosion of style sheets and solved that problem by condensing them all. But solving one problem produced another. **Problem**: With all the shifts in mid-testing stream, finding the status of what had been tested with the particular environment and what was yet to be tested became a monumental task that was no longer manageable with just Word documents. On the other hand, buying a product was not a budgetary possibility. **Solution**: The problem of tracking all this testing was solved by developing an MS Access Test Case Management system, managing both test cases and execution outcomes. **Lesson learned – six**: Test planning, test management, test maintenance, and test statusing all require a process and infrastructure from the beginning. Without the infrastructure, ongoing testing activities become at least an order of magnitude harder to manage and status.

There are many differences between WAP and HTTP but the single problem that caused us enough problems to transfer our efforts to working with browsers that used HTTP was the amount of data that can be easily transmitted for one screen. Several of the browsers had a card size of 1600 bytes. Deducting the overhead for formatting information and structures (WML tags, buttons, links), there was not adequate space available left for the amount of data our product was requried to display. Cards can be stacked to give more information, but that is adds complexity for maintenance as well as for developing and testing. **Problem:** How do you overcome the space limitiations imposed by WAP? **Solution**: Using HTTP browsers wherever possible seemed an easier solution. Using an HTTP browser had the additional bonuses of being more familiar to the developers, allowing more data to transfer to the device in one unit, and being more complete and standard than the various WAP browsers used for prototyping. **Lesson learned – seven**: Use what works the best in the current situation, even if it is not the newest, shiniest technology.

Problem: The WAP vs HTTP choice was not always a choice. There are some hand held devices whose only browsers were WAP based. In those cases, we had to use WAP/WML, limit the amount of data sent to the amount that WAP would handle, and test entirely again due to the new code involved. **Solution:** Just do it. **Lesson learned – eight:** Sometimes testing is just tedious, repetitive work! ;>} This is especially true when you have no automation tools. While automation is not a silver bullet, when trying to test many environments with the same tests and limited resources, automation tools are really nice to have.

Screen real estate is extremely limited

Users complain if they are stuck with a fifteen inch screen on their desk tops, so it is amazing that they want to use a device with a screen that is typically two inches by two inches! From a design perspective, that two by two screen is especially challenging. The screens that were being "reproduced" for our wireless product use were complex and took a lot of screen real estate on the seventeen inch desktop screens. For this reason, it was an arduous process to break the functions down into user friendly chunks that would fit on the small screen and still get the user to their data in an average of three clicks. In spite of all the work that went into the design, it was usability testing that caught the major user annoyances on the small screen. For example, on a large screen with large amounts of data available and easily accessible, it is not as important for the user to know exactly how long the result set is. However, when each batch of five cases displayed on the screen involves a round trip to the application server, it is absolutely imperative that the user know the total length of the result set that they are trying to traverse. For this reason, test lobbied for and got result and list length indicators on all list display screens. Another example, of the usability testing impact, were add shortcuts that allowed the user to get to their data in three clicks from just about anywhere in the product. **Problem**: How can the user perception of the product's usefulness be enhanced inspite of the small display screen? **Solution** : Test expended almost as much effort making the product easily usable as they did testing for the quality assessment of the product. **Lesson learned – nine:** Usability testing needs far more attention in the wireless arena, due to the small screen. User friendliness is a larger factor in the user's ultimate perception of the product than in many other types of software products.

Lessons Learned Reviewed

1. Including test (and others) up front through design review produces a more robust design that survives the test phase better and cuts the time and cost of development.
2. Having a carefully thought out template defined before teams split out to design in parallel saves time and re-work.
3. Start development and testing with long term plans for handing the product's complexity, versioning and eventual maintenance.
4. Testing for usability means looking past the design both for the wireless product and the desktop product
5. Do not forget the negative condition testing.

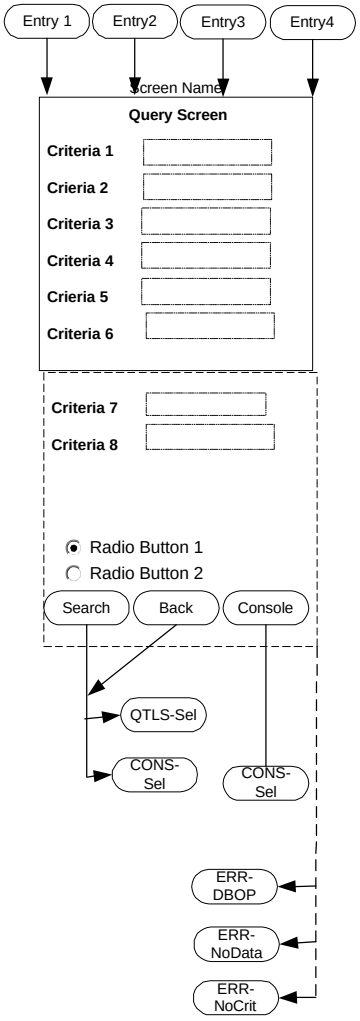
6. Test planning, test management, test maintenance, and test statusing all require a process and infrastructure from the beginning.
7. Use what works the best in the current situation, even if it is not the newest, shiniest technology.
8. Sometimes testing is just tedious, repetitive work! ;>}
9. Usability testing needs far more attention in the wireless arena, due to the small screen.

Conclusion

In looking at the problems we encountered, our solutions for them and our lessons learned, the conclusion that was drawn was that when testing anything, the problem may stem from the technology encountered and the solution may involve the technology of the product being tested, but, the underlying testing principles are the same regardless of the technology. This may apply only because we were doing black box testing. However, in my opinion, even with much more technically oriented testing like tracing packets through protocols, when the actual root cause analysis of the problem is analyzed, problems encountered stem from one of the basic good testing practices being violated. I proposed and started writing this paper expecting to share some insight into testing wirelessly and the insight that I found was that good testing practices are basically good testing practices regardless of the technology to which they are applied. There is no testing substitute for good analysis and planning applied through solid testing principles. Apparently that hasn't changed since the dawn of computers!

Appendices

Appendix A: GUI sample design page



Purpose: Text describing the general purpose of the screen and an overview of the functions possible on this screen.

Navigation Section:
All the possible screens that can reach this screen are mentioned. Each should have a bubble with an "in" arrow at the top of the picture. All possible screens that are reached by this screen are mentioned. Each should have bubble with an "out" arrow at the bottom of the picture.

Static Data Description Section:
This section describes all the static data on the screen.

Field Description Section:
Field by field description of the input and data display fields on the screen along with any information needed to validate the data in tables or elsewhere if data validation needs to be done.

Button and Link Section:
This section describes each button and link on the screen and its function and navigation effects.

Exception Handling Section:
List of all the exception handling that will occur and what the message will be.

The screen navigation into and out of the screen are portrayed by the ovals with the screen name in them and solid arrows.

Ovals with dashed line arrows are exception handling dialogs that will appear.

·The screen size should be the actual device screen size to make the design more realistic.

Appendix B: Glossary

Term	Definition
Application Servlet	In our case, Java code resident on the web server that services the function requests from our screens.
Button	An icon on the screen that looks like a button, which when selected, causes an action.
Extensible Markup Language (XML)	A language that facilitates document exchange between systems by describing documents in a system independent manner.
Extensible Style Language (XSL)	The file that defines the way the data will be displayed on the device.
Graphical User Interface (GUI)	The visible part of the product presented to the user.
Hypertext Markup Language (HTML)	A tag language that is used to display data on Web pages.
Hypertext Transfer Protocol (HTTP)	The protocol used to transfer data to and from Web browsers and some wireless browsers.
Link	Portion of the screen display, which when selected navigates the user to a new location in the product. Typically represented by underlined text.
N-Tier architecture	Architecture in which the business code is in one or more layers between the client and the database.
Shortcut	A Link, button or key combination that navigates the user directly to a particular point in the product.
Wireless Application Protocol (WAP)	A protocol defined by the WAP forum for standardizing wireless usage in the U.S.
Wireless Markup Language (WML)	A markup language designed specifically to work with the wireless application protocol.
Wireless Device	These are devices such as a Palm Vx, a Nokia phone or a Compaq Pocket PC which can be used to access the Internet and intranets wirelessly.
Wireless Gateway	Software that translates from WAP to HTTP in one direction and from HTTP to WAP in the reverse direction.
Wireless Markup Language (WML)	A markup language similar to XML used with the wireless protocol.
Wireless Service Provider (WSP)	The company that provides the wireless access to the Internet.

Appendix C: References

Books:

Arehart,Charles, et al., Professional WAP,Wrox Press Ltd.,Birmingham, UK, 2000.

St. Laurent, Simon. XML: A Primer, MIS:Press, Foster City, CA, 1998.

Web Sites:

802.11 Working Group for Wireless Local Area Networks
802.15 Working Group for Wireless Personal Area Networks
802.16 Working Group for Broadband Wireless Access Standards
http://standards.ieee.org/wireless/overview.html

W3C was started in 1994 to lead the Web to its full potential by developing common protocols that promote its evolution and ensure its interoperability.
http://www.w3c.org/

A portal address for broad categories of information concerning wireless communication
http://www.links2wireless.com/standards_protocol.htm

Geraldine Conley

Geraldine Conley has worked in the computer field since 1965. Conley has played many roles in several companies over the years, including Programmer, IMS DBA, AI Knowledge Engineer, Software Test, and Software QA. Conley is currently a Software QA Manager for a startup. Conley's education includes; an A.A, B.A., B.S. and she has completed the USCS extension course certificate in Software QA.