Quantification of Human Factors and Quality Control for Wireless Development

Author: Reza B'Far
Acknowledgements and Contributions by: Roger Richards and Stephen Ditlinger

Abstract

Development of wireless applications for mobile devices (such as cell phones and PDAs) presents software engineers, quality assurance engineers, and human factors engineers with a new breed of challenges. One of these challenges is to make science from something that is art today: design and quality assurance for a user interface. This paper intends to outline a suggested technique for quantifying design and quality assurance for wireless development. While the specific example used here is a WAP application, the same set of principals and mathematical models can be used to quantify other types of mobile and wireless applications. The model suggested in this paper can be used as the specifications for a tool which evaluates various user interfaces for devices, particularly mobile and wireless devices.

Introduction

Before embarking on a comprehensive discussion of design and analysis of the human interface for wireless development, let us first consider why such an effort is so different from human interface design for a more typical Web application. Additional problems faced by user interface designers in wireless application development are:

- **Limited Bandwidth**: A wireless device typically has much less bandwidth available for transmitting and receiving data than a wired device.
- **Intermittent Connection**: The connection to a wireless device is typically unreliable. A persistent point-to-point connection is difficult if not impossible.
- **Limited Battery Life**: A wireless device is typically (also) a mobile device. Since mobility dictates compactness in size, and since there is no wired power connection, batteries are the only means of power supply. Even the longest lasting batteries offer a very limited amount of power.

eBuilt
2652 McGaw Ave., Irvine, CA 92614

- **Limited Memory on Client Device:** Once again, because of the mobile nature of wireless devices and their requirements to remain a small size, room for memory is limited. Memory is also limited by the available power source (batteries) on the device.
- **Limited CPU:** Because of the size of the devices and the battery life, processing information on the device is very expensive. Very few operations should be performed on the device and they should be only performed where there is strong justification for them.
- **Limited User Interface:** A keyboard and/or a mouse are normally not available for a wireless or a mobile device. Also, the display is almost always very small. This makes viewing and data entry more difficult.

In this paper, we consider a three-tier architecture where the client browser, Web server side operations, and database back-end operations each form a separate tier. Such architectures have proven to be scalable and reliable. When designing a three-tier architecture for a wireless application, one quickly discovers that there is no significant difference in database and middle-tier design between a wireless application and any other types of Web-based applications. The aforementioned problems are almost completely confined to the client tier. In other words, if the system is designed so that business logic is decoupled from presentation, and that business logic is encapsulated in the middle-tier, the most significant task in enabling users to access existing functionality through wireless devices becomes developing a new GUI layer. While this is the primary objective, it is also important to remember that various devices offer functionality previously unavailable on PCs (such as proximity information and history of path traveled). Therefore, there are additional functionalities whose additions to the system might be desired based on the device set supported.

The prevailing method for allowing various devices to access the same content is by pushing content into XML instead of HTML. Using Extensible Stylesheet Language (XSL), the XML formatted content can be transformed for presentation into HTML or other formats. This is altogether a separate discussion, which we will leave to the user to investigate more thoroughly. It is assumed, in this discussion, that content provided by the middle tier is in an XML compliant format. The consideration of methodologies for the user interface in this paper is based on the XML content.

Initial Design

Before the actual human interface design process begins, it is important to set some boundaries on development and prepare some tools for the development and quality assurance process. Therefore, the following steps must be followed:

1. Device Set Selection

   Before designing the interfaces, it is important to select the primary set of devices that the application intends to support. This is particularly important as devices and platforms vary greatly in CPU size and data entry mechanism (PDAs use a stylus and touch-screen as an input mechanism, while WAP compliant cell phones support telephony functions through WTA 1.2, etc.). Once a device type is identified, it is also necessary to specify a device manufacturer and, perhaps, even a model number. For example, we can narrow our support to Palm PDAs and Nokia cell phones as platforms. Further, we can specify that the application will support Palm VIIx and Nokia 7110 cell phones.

2652 McGaw Ave., Irvine, CA 92614

2. Emulation of Selected Devices:

   Once the subset of devices to be supported by the particular application has been determined, both the designers and the quality control engineers must obtain emulators for the selected devices. The first step of unit testing and quality assurance testing is testing by using the emulators. For our example, we can use Palm emulators that are available from the Palm Web site. Nokia has a WAP development tool kit that has a full-blown IDE for WAP, including an emulator, which allows support for multiple Nokia cell phones and PDAs.

Designing the First Set of Interfaces

Once the devices to be supported by the application have been identified and the emulators for those devices have been gathered, it is time to start the design process.

> **Note:** We will forego a detailed discussion of the process of requirements gathering and use-case creation with the assumption that the reader is already familiar with those processes.

Subsequent to development of the use cases, it is time to begin designing the first set of user interfaces. We will use a Ticket Purchasing application as an example.

Gathering the Use Case Scenarios

Let us assume that we have two use cases for our example (as the subset of all of the use-cases for the application):

1. Use Case 1, User Log In: display a user name and password to the user.
2. Use Case 2, Main Menu: display a main menu that will allow the user to perform basic navigation to the other parts of the wireless application.

Identifying Display and Input Elements

The biggest concern is the limited user interface. Therefore, we have to clearly define what the displayed elements and the input elements for each step of each use-case should be. Once again, for our example:

Use Case 1

- *Data Elements*: User Name, Password.
- *Actions*: Cancel, OK

Use Case 2

- *Data Elements*: Menu Items—Search for Events, Purchase Tickets, Reserve Tickets, My Events Calendar, and Log Out.
- *Actions*: Scrolling using Select Arrows, Cancel, OK, and Back.

eBuilt

2652 McGaw Ave., Irvine, CA 92614

There are three types of basic elements that can be identified:

- *Data Elements*: These elements are elements that display data or ask the user for some text input to the device.
- *Action Controls*: These controls are those keys (or buttons on the screen to be stabbed, or else as might be applicable to the device) that cause navigation to a different display screen (in WAP, this is a card—in cHTML, it's a different cHTML file, and so on)
- *Navigation Controls*: These controls are those that allow the user to scroll up and down and/or navigate within the same display screen (in WAP, this is a WML card; for i-mode, it's a different cHTML file, and so on)

Creating XML Content Based on Display and Input Elements

Since the content has been displayed in many different ways, the next step is to create all content in XML. Once again, the discussion is outside of the scope of this paper; however, it is important to remember that to support a variety of devices, the best architecture is one that uses XSLT to transform raw XML content to formatted XML content usable by a given device based on the characteristics of the device. For our example, our raw content might look something like this:

Use Case 1

```
<?xml version="1.0" ?>
<myxmlformat>
   <page number="1">
      <input type="text" name="username" />
      <input type="password" name="password" />
      <action type="nextpage" value="2" />
   </page>
</myxmlformat>
```

Use Case 2

```
<?xml version="1.0" ?>
<myxmlformat>
   <page number="2">
      <select type="numbered">
         <option value="Search">Search For Events</option>
         <option value="Purchase">Purchase Tickets</option>
         <option value="Reservations">Reserve
         Tickets</option>
         <option value="MyEvents">My Events</option>
         <option value="LogOut">Log Out</option>
      </select>
   </page>
<myxmlformat>
```


eBuilt

2652 McGaw Ave., Irvine, CA 92614

We have created our own XML format that merely represents the data and some generalized representation of the behavior of the user interface. There is no client-specific information (browser or micro-browser related) about where a text element should be placed or how it should be presented.

Creating Formatted XML for the Supported Devices

Once the XML content has been developed, we need to create a mechanism that creates different types of content for different types of clients (Cell Phones, PDAs, browsers, micro-browsers, and so on). For example, we might need to create a variety of HTML pages for different browser types, or we might have to produce a series of WML pages, each created for a different device. This translation from XML to other formats is done by creating an XSL for every variation of the transformed XML. However, it quickly becomes clear that the permutations of the XSL increase exponentially to support more and more devices. Dynamic generation of such content does not scale well.

Three possible solutions are discussed in the next section:

1. Pre-generation of all static content.
2. Using scripting languages, such as JSP, to process the XML instead of using XSLT.
3. Generalization of some of the possible permutations of the user interface, per device families.

Iterative Quality Analysis and Redesign

After the initial phase of the design, we will have a set of possible permutations for the user interface. For our example, two of those permutations (for use case 1) might look something like this for WML:

Use Case 1

Permutation 1

```
<?xml version="1.0"?>
<!-- created by WAPtor (http://www.waptop.net/) -->
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
   <card id="card1" title="USERNAME" >
      <p>
      User Name:<input type="text" name="username" />
      </p>
      <do type="accept">
         <go href="#card2" />
      </do>
   </card>
```


eBuilt
2652 McGaw Ave., Irvine, CA 92614

```
      <card id="card2" title="PASSWORD" >
         <p>
         Password: <input type="password" name="password" />
         </p>
         <do type="accept">
            <go method="post"
            href="http://wireless.ebuilt.com/servlets/Login" >
               <postfield name="server_username"
               value="$(username)" />
               <postfield name="server_password"
               value="$(password)" />
            </go>
         </do>

      </card>
   </wml>
```

Permutation 2

```
<?xml version="1.0"?>
<!-- created by WAPtor (http://www.waptop.net/) -->
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
      <wml>
         <card id="card1" title="LOGIN" >
            <p>
            User Name:<input type="text" name="username" />
            Password: <input type="password" name="password" />
            <do type="accept">
               <go method="post"
               href="http://wireless.ebuilt.com/servlets/Login" >
                  <postfield name="server_username"
                  value="$(username)" />
                  <postfield name="server_password"
                  value="$(password)" />
               </go>
            </do>
            </p>
         </card>
      </wml>
```

As you can see, though there is very limited input and only one screen here, we have already started down two different branches on our navigation tree. This navigation tree will be very large for those applications supporting a large set of client types (device types and browsers). And they grow by factorials every time there is a new XML file (in our case a WML file).

eBuilt

2652 McGaw Ave., Irvine, CA 92614

Therefore, we need to deal with the problem of scaling in two fronts: development of XSL files and run-time content generation. Developing new XSLs for every device is impractical as development cost is prohibitive. Also, as the number of XSLs grows and the rules on each XSL become more complicated, the XSLT compiler becomes slower.

Content Pre-Generation

One of the solutions to the problem is to pre-generate the transformed XML on a batch mode. Based on some event triggered by change in the original XML content or using some period, we can generate new content for each device based on the various XSLs. This option solves the latency caused by the XSLT to generate transformed XML; however, selecting which transformed XML should be served to which device remains a problem.

Generating XML Using Server-side Scripting

To reduce the load on the XSLT when generating the XSL, it is possible to put the complex logic in a scripting language that wraps XML. This can be done, depending on the application, before or after XSLT processing. This allows any complex logic involving selection of the rules for XSL templates to move into a server side scripting language, such as JSP, thereby making the process more efficient.

Selecting the Most Effective Set of Interfaces

While the two methods described above reduce the load on the XSL generator at run-time, the number of different user interfaces and the development of various XSLs grows problematic as the number of devices to be supported by the application increase. Therefore, we need to limit the number of possible XSLs by generalizing the user interfaces available to the devices. In other words, we can assign one set of user interfaces to a group of devices instead of a single device. To do this, we need a method to compare and evaluate the various user interfaces. In a way, what we are attempting to do is to compress the information content of the user interface, both in data display and interaction and in intra-display navigation. This compression, as any other compression method, can be lossy or lossless. Since loss is introduced in “generalizations,” and since we will be generalizing groups of devices into categories or families of devices, our content compression of the user interface will be lossy.

Now, let us leverage some knowledge from compression in information theory, namely the concept of a Huffman Coding Length. One of the easiest forms of data compression, Huffman Coding compression allows for a very simplistic method of assigning numbers according to the possibility of occurrence of each element in a system. For example, let us look at a system where all the content is composed of letters {A,B,C,D}. Let us assume that the possibility of the occurrence of each of these letters is, respectively, { 0.1, 0.5, 0.3, 0.2 }. Then, the most compressed representation of the data using 0’s and 1’s would be { 01, 0 , 1 , 10 } and/or { 10, 0 , 1 , 01 }, assuming variable length bytes (we will not discuss how the bytes are constructed here since compression for variable length content is complicated). Shortest Huffman Coding lengths are, in order: B,C,D,A and/or B,C,A,D

eBuilt

2652 McGaw Ave., Irvine, CA 92614

(depending on negative or positive logic). Therefore, we need to first quantify the set that describes our content, and then we need to compress it.

In this manner, we will introduce a set for the interactions between a user and a device. Then, we will introduce a grading system that can be used to determine the degree of loss in compressing the information exchanged between the user and the device. Our grading system will allow us to quantify the user friendliness of each set of user interfaces. Note that our grading system will use variables $\{P_1, P_2, \ldots, P_n\}$, so that we can dynamically change the grading system based on the family of devices. For example, we might assign the grades to be {1,2,0,5,….10} for all cell phones and {2,5,9,10,….15} for all PDAs.

As previously mentioned, we are concerned with two client interaction problems: display/input of data and navigation. These problems encapsulate the variables in our set. Please note that the set we have defined here is only a subset of a superset that might include other user interactions with the device, such as voice, touch movements, and so on.

We will then define our subset for user interaction content components as:

Scrolling: (P_1, P_2)

Scrolling up and down (or side ways if the device allows) is very cumbersome. Scrolling more than one page, in addition to the viewable page (for a total of two pages), causes even more trouble for the user. For our grading system, we will assign P_1 points any time the user has to scroll and P_2 points every time the user has to scroll more than one page. Therefore, if the screen has four lines and there are eight lines of content, we add P_1 points to the point total of that screen. If the screen has nine lines of content, we add P_1+P_2 points to the point total of that screen.

Text Entry

There are various types of text that the user can input into a device. We rate the different types here:

Repeating Keys (P_3 , P_4)

Certain text, such as letters “s” and “k,” require multiple pushes from the same button. This is cumbersome since the time delay allowed between the key pushes to type two consecutive letters and the delay allowed to get to the a letter that requires multiple pushes are different. We will assign a grade of P_3 any time a multiple push character is required and a multiplier of P_4 as a coefficient for the number of times that button must be pushed (twice, three times, and so on).

Alpha Entry (P_5)

Depending on the device, certain characters such as “(“ or “$” might require navigation to a different screen. Most of the time, this is indicated by a prompt allowing the user to navigate to an area marked “ALPHA.” Such characters cause extra confusion and force the user to do

extra navigation. We will assign P_5 to any character that requires navigation to another screen.

Numeric Entry (P_6)

Numeric data entry, on most devices, is different than text entry. For example, a particular environment for a cell phone might allow for overriding the text associated with each key and allow for using the keys for numeric entry (this can be done using input masks and WMLScript in WAP). We need to also consider those devices that might allow only numeric entry. We will assign P_6 to any numeric character.

Stabbing (P_7)

In devices, such as the Palm Pilot, a pen (Stylus in case of the Palm—a spear) is provided to "stab" certain buttons on a touch screen. This pen is used for data entry, as well as stabbing different buttons on the screen (the touch-screen aspect and the provided stylus are actually independent, but for our discussion here, we will assume that the functionality is combined). We will assign P_7 to any singular use of the pen (or any Stylus type device) for interaction with a device.

Inter-Page Navigation

Forward Navigation Using Device Buttons (P_8)

It is possible to use various buttons (such as the left top button on Nokia phones used as OK) for navigation. Various devices map the available keys to different navigation rules. We will assign P_8 to use any device specific navigation.

Forward/Backward Navigation Using Device Buttons (P_9)

Because of the display limitations (for example on cell phones), the user might be required to go into a submenu first, enter or view some data, and then return to the parent menu. We refer to that as forward/backward navigation. This is particularly annoying because it causes confusion for the user. We will assign P_9 to forward/backward navigation using custom buttons on the device.

Forward Navigation Using Menus (P_{10})

This is the same type of navigation suggested in Forward Navigation Using Device Buttons except that the device buttons are not used. This refers to navigation using a menu system with radial buttons and/or selecting a menu item on the screen. We will assign P_{10} to forward navigation.

Forward/Backward Navigation Using Menus (P_{11})

This is the same type of navigation suggested in Forward/Backward Navigation Using Device Buttons except that the device buttons are not used. It refers to forward/backward navigation with radial buttons and/or selecting a menu item on the screen. We will assign P_{11} to forward/backward navigation

It is obvious that our quick analysis here was done with a bias for cell phones. It is very important to remember that what we have defined here is a methodology and a general approach. This paper, by no means, intends to define all of the independent variables for all devices—this is something that will remain application-specific. There are many device families that display information in different ways and interact with the user in many different ways. So long as the methodology is followed per device family and a set is clearly defined for each, the methodology will remain valid and will apply.

Therefore, the subset that we have defined for our application is:
$\{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}\}$

Also, we can select multipliers which allow a different "weight" for each of the independent variables. This is to allow modeling the fact that not all actions are equally as easy or as difficult. For example, using the Stylus on the Palm platform to stab an icon on the touch sensitive screen can be deemed easier than pushing a physical button on the Palm. Both account for 1 action, but their cost is different.

Now, let us apply this methodology to our example.

Application of Methodology

Now, let us evaluate the sets for our use cases. Here, we will only give you the step-by-step evaluation for Nokia 7110 of use case 1. You can extrapolate the results for yourself for the other use case and for other devices. We will use "sditlinger" as the user name and "rrichards" as the password.

Key Sequences:

1. Press Option (P_{10}).
2. Select Edit (P_8).
3. s requires four key presses, d requires one key press, etc. Total key presses for "sditlinger" is 23 (P_3).
4. s, i, l, and some other letters require multiple key presses on the same key. This happens seven times (P_4).
5. Once the user name is entered, Press OK. This causes navigation to a previous screen (P_9).
6. Select Edit (P_{10}).
7. Press Clear (P_9) five times to clear the user name that was put in previously.
8. Total key presses for "rrichards" is 23 (P_3).
9. Multiple key presses happens seven times (P_4).
10. Press Ok. This causes navigation to a previous screen (P_9).

eBuilt
2652 McGaw Ave., Irvine, CA 92614

11. Press Options (P_8).
12. Press Down–Arrow once (P_8).
13. Select OK (P_9).

There is no numeric-specific data entry, scrolling, or stabbing. Therefore, P_1, P_2, P_5, and P_6 are 0. Though one can represent the final result as a combination of different functions of the independent variables, we will assume that the "weight" of that occurrence of each instance of each independent variable is the same for the final result. In other words, we will assume that pressing the OK button has the same cost whether it is the first step of navigation or the last step of navigation. Though this assumption is not necessarily true, it will give us a very close linear approximation to what might be true.

Based on the above, we obtain the following for Permutation 1 of our code for Nokia 7110:
$\{P_1, P_2, P_3, P_4, P_5, P_6, P_7, P_8, P_9, P_{10}, P_{11}\} \rightarrow \{0,0,46,14,0,0,0,8,3,2,0\}$

Once again, we'll make an assumption of linearity between the different orthogonal variables. This means that we'll assume that the costs of all the independent variables are equal (the cost of pushing the OK button and pushing a text button is the same).

This will let us come up with an average length that is the sum of the value of all variables divided by the number of variables. Therefore, the average length for our example is 6.36 for permutation 1 for Nokia 7110 (total score is 73 and there are 11 independent variables, yielding an average length is 6.36).

Obviously, if a large number of devices in each family are to be supported, this process needs to be automated with a tool. The tool can be used to reveal the flexibility of the methodology to the user interface designers, as well as the quality assurance engineers.

It is also important to remember that the amount of code (number of characters) written in WML, as well as calls to WMLScript, needs to be taken into account. Minimizing code is crucial to maximize use of the device. Code efficiency considerations can be treated as other independent variables in our model or can be considered separately. Here we have decided to treat them separately.

For our example, we tested the following devices with the respective results.

Permutation 1 (585 Code Characters):

Device Name	Use Case 1 Score	Use Case 2 Score	Total Score	Device Family
Nokia 7110	6.64	1.82	8.46	Nokia Phones
Phone.com Emulator	5.64	1.82	7.46	Phone.com Browser

eBuilt
2652 McGaw Ave., Irvine, CA 92614

Permutation 2 (496 Code Characters):

Device Name	Use Case 1 Score	Use Case 2 Score	Total Score	Device Family
Nokia 7110	5.73	1.82	7.55	Nokia Phones
Phone.com Emulator	5.64	1.82	7.46	Phone.com Browser

Summary

This paper was an attempt at quantifying the interaction between a user and a device. Such quantification is most applicable to devices that have limited capabilities, such as today's wireless and mobile devices. Once the problem was defined, we approached the solution by looking at the interaction between the user and the device as content that can be represented as a composition of a set of orthogonal and independent atomic interactions, such as entering text. First, we saw that we can use XSL, scripting languages (such as JSP), and good design decisions for a path to a better system. Later, we formalized a methodology to select the best set of screens based on this information.

Also, it is important to remember that business requirements must be considered carefully in the original design of the set of user interfaces. If business requirements of the applications make no sense (for example, e-mail is accessed through purchase ordering menu), the business process flow is disrupted. Business logic and flow must be considered before the first set of user interfaces are selected and put through the selection process suggested in this paper. Therefore, the steps of development and quality assurance should represent an iterative approach, as suggested by the following steps:

1. Defining application requirements, use-cases, and so on (typical software development requirements gathering process).
2. Defining device families to be supported for the application.
3. Defining the device sets within each device family to be supported for the application.
4. Defining an orthogonal (independent) variable-set that describes the domain for the interactions of the user with the device families.
5. Designing the first set of user interfaces based on the business rules.
6. Applying the orthogonal variable-set to create a grading system.
7. Using the grading system to evaluate the various possible permutations of the user interface per device family.
8. Creating a set of XSLs for each device family. Custom XSLs might be created for specific devices whose use might be inordinately higher than other devices (at least one order of magnitude).

The most critical argument to be made is that all user interaction with a system is information. In order to come up with the single most efficient user interface, we need to choose the most efficient

method of transmitting the information. This will speed up the user interaction with the system, thereby producing a more pleasant experience.

About eBuilt:

eBuilt designs and builds industrial-strength e-businesses. The company provides custom applications development and integration services to build reliable, robust and scalable Web infrastructures for new and existing e-businesses.

eBuilt Wireless Development Group is an internal group at eBuilt, Inc. focusing on research and development of wireless technologies.

About Author and Contributors:

Reza B'Far and Stephen Ditlinger are senior software engineers at eBuilt, Inc. Roger Richards is a project manager at eBuilt, Inc. They are all members of the eBuilt Wireless Development Group.

Reza B'Far, the author of the paper, has an academic background in communication systems and work experience in commercial software development. He has worked on 3-tier systems based on J2EE technologies, other Java development, various web based technologies, image processing, reporting systems, and data mining systems.

Roger Richards has an academic background in computer engineering from University of Illinois. He has developed various applications on embedded and client-server systems. Since then, he has managed web based solutions development for numerous projects.

Stephen Ditlinger has an academic background in systems engineering and optimization. He has worked on 3-tier systems based on J2EE technologies and other Java based technologies. He teaches Java and Object Oriented Design courses at a local state university.

2652 McGaw Ave., Irvine, CA 92614