Usability and Reusability

Chris Ambler, Head of Testing Architecture and Strategies at Newell & Budge, on usability testing with scripts

In today's fast moving technological climate, it is safe to say that the only constant is change. Technology moves nearly as fast as people's ideas and new ways of doing things are created every day. In the software world, this means that Graphical User Interface (GUI) front ends are always under review and change. A changing GUI can have a massive impact on the way the business is run. Users that cannot use their 'tools' will find other more efficient (but less effective) ways of carrying out their daily tasks. As testers, this causes a number of problems when it comes to writing re-usable usability scripts. A balance must be achieved between 're-inventing the wheel' and utilising old assets every time there are changes to the front end. During development, the main goal of a usability test is usually 'diagnostic' and used to find out what is performing correctly and what is not working well, so that development can continue with what is working and the developers and system testers can fix what needs more work. The earlier this is done and the more iteratively, the better.

Once development and system testing is complete, it can be passed to the users/testers. The most important thing to remember at this stage is usability testing is not designed for functional diagnostic testing, it is more of a verification process to ensure that users can complete a task successfully and it is fast enough to satisfy them, the paths they take are perceived to be efficient enough for them and that they do not have any problems or get confused anywhere. It ensures that the application or system ergonomic qualities and overall end–to–end interaction is satisfactory and follow the Jacob Nielson usability heuristics. These heuristics are:

1 **Visibility of system status:** The system must always keep the user informed about what is going on. This is done by feedback within reasonable period of time. From a testing point of view, it is necessary to understand, define and quantify 'reasonable'.

2 **Match between the system and the real world:** The system must speak the users' language, using familiar words, phrases and concepts. It must avoid using system-oriented 'jargon'. It is necessary to follow real-world conventions as much as possible, making information appear in a natural and logical order.

3 **User control and freedom:** It is a fact of life that users will often make mistakes. This creates the necessity for needing a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended procedure. It is important that the application supports 'undo' and 'redo' facilities.

4 **Consistency and standards:** Users should not have to wonder whether different words, situations, or actions mean the same thing. The specified platform conventions need to be followed, allowing a user to be comfortable with the 'look and feel' of the system.

5 **Error prevention:** Even better than good error messages is a careful design, which prevents a problem from occurring in the first place. When error messages occur, they need to be necessary, or alternatively be avoided by better design.

6 **Recognition rather than recall:** Objects, actions, and options need to be visible. The user should not have to remember information from one part of the dialogue or process to another. Instructions for use of the system should be visible or easily retrievable whenever the user requires them.

7 **Flexibility and efficiency of use:** The system needs to be as flexible as possible for differing levels of users. It is sometimes useful if experienced users can tailor frequent actions. Accelerators, unseen by the novice user, may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users.

8 **Aesthetic and minimalist design:** Dialogues should not contain information, which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility to the user.

9 **Help users recognise, diagnose, and recover from errors:** Error messages should be expressed in plain language and contain no codes (unless as part of an error message that may help the troubleshooting process). They must also precisely indicate the problem, and constructively suggest a solution.

10 **Help and documentation:** Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any information should be easy to search, focused on the user's task. It must list concrete steps to be carried out, and not be too large.

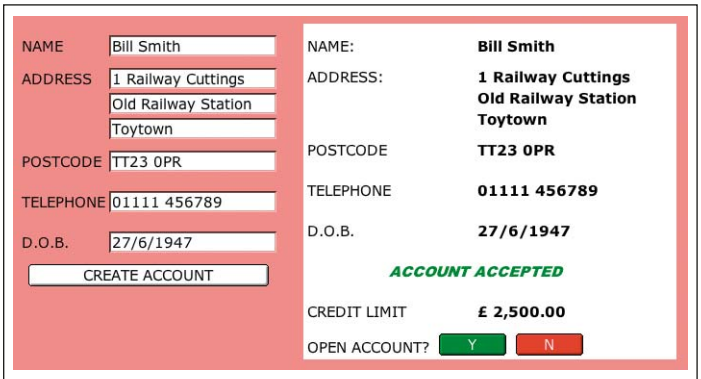

Figure 1: Account creation dialog

Let's look at a very simple example. A sales system has a dialog for creating a customer account (figure 1).

A system test for this screen would go into the detail of each input field size, check the

formats, validate the credit checking interface, check the screen outputs etc. and would need a very detailed, step by step system test script with detailed expected results. Any changes that are made to the screen will need detailed changes to the test script and test data. This will have major impact on configuration management and version control.

As usability testing is defined at a higher level, a process scenario can be created and each of the usability heuristics can be assessed during the process. This usability test script would look like the one shown in figure 2.

This test has raised a number of 'potential' defects. These defects may or may not be important to the users, given the context in which they are working. I am sure every reader will come up with a different set of 'issues' with this example!

Usability testing needs to be more 'subjective' rather than 'objective'. This creates the need for the tester to understand the process under test and be able to improvise the additions to the screens (as long as they are documented). The major changes will occur in the test data sets to ensure repeatability and again, the experienced user/tester needs to control these and document all changes.

If a change was made to the account creation screen, for example an 'ANNUAL SALARY' field was added, this would not change the usability test script as the scenario described has not changed. The only change would be to add a test data item to the test data set that would allow the user/tester to enter an annual salary. This is only an extra column to a test data spreadsheet or test data list and can be controlled easily.

It is impossible to completely 'future proof' usability test scripts because there may be more complex changes to the screen or changes to the process itself. The best you could ever hope for is to design them in a way that allows changes to be made easily. This is achievable as long as the key rules are remembered:

- usability testing is not about proving functionality: it's about verifying tasks
- iterative testing is best – knowledge of the business processes is key
- test scripts should be built around scenarios
- communication between users, testers and developers is paramount
- the secret is not to have detailed scripts
- test data needs to be updated and managed

Developers have been writing code and testing it against scenarios for years, so it is a tried and trusted technique and if the relationship is good enough why not communicate with the developers and look at linking the development changes to the usability test changes? Working with the developers and the users on the required changes to the GUI allows the tester to both understand the changes and inform the users (and developers) what the impact of those changes will be. It is possible to link these changes to business risk and business priorities, but that is another story… PT

USABILITY TEST SCRIPT – CREATE ACCOUNT **TESTER:** _____

Process Scenario:

The salesperson enters the customer details on the left hand side of the screen. On completion of the details, the 'CREATE ACCOUNT' button is clicked. The system then populates the box on the right, carries out a credit check on the applicant and accepts or rejects the applicant. If the 'ACCOUNT ACCEPTED' is shown, a credit limit is displayed and the 'OPEN ACCOUNT' 'Y' and 'N' buttons are available. If 'ACCOUNT REJECTED' is shown, then no credit limit is displayed and the 'OPEN ACCOUNT' 'Y' button is greyed with only the 'N' available. The salesperson then clicks the relevant 'Y' or 'N' button

Test Scenario:_____ Test Data Set: TDS001 Application Version: v2.01.a Environment: Test07

Expected Result : Successful account creation

Heuristic	Comments	Pass/Fail
1 Visibility of system status	There is no visible means of showing the system status other than the usual 'timer'	FAIL
2 Match between the system and the real world	All the terms used are consistent and have real world meanings (although D.O.B. could be open to misinterpretation)	PASS
3 User control and freedom	There is no obvious 'emergency exit'. There is no way to exit the screen until the process is complete, either successfully or unsuccessfully.	FAIL
4 Consistency and standards	Consistency with the rest of the application is acceptable	PASS
5 Error prevention	There is no error prevention available to this screen	FAIL
6 Recognition rather than recall	Recognition and recall is acceptable	PASS
7 Flexibility and efficiency of use	The screen is flexible and efficient in accordance with its use	PASS
8 Aesthetic and minimalist design	The screen is not 'cluttered' and all information shown is pertinent to the process under test	PASS
9 Help users recognise, diagnose, and recover from errors	Errors can only be identified at the end of the process at 'ACCOUNT REJECTED'	FAIL
10 Help and documentation	There is no help documentation	FAIL

Figure 2: Usability test script