A Tale of Two Studies

Judy Bowen[1] **Steve Reeves** [2] **Andrea Schweer** [3]

(1,2) Department of Computer Science
The University of Waikato,
Hamilton, New Zealand
Email: jbowen@cs.waikato.ac.nz, stever@cs.waikato.ac.nz

3 ITS Information Systems
The University of Waikato,
Hamilton, New Zealand
Email: schweer@waikato.ac.nz

Abstract

Running user evaluation studies is a useful way of getting feedback on partially or fully implemented software systems. Unlike hypothesis-based testing (where specific design decisions can be tested or comparisons made between design choices) the aim is to find as many problems (both usability and functional) as possible prior to implementation or release. It is particularly useful in small-scale development projects that may lack the resources and expertise for other types of usability testing. Developing a user-study that successfully and efficiently performs this task is not always straightforward however. It may not be obvious how to decide what the participants should be asked to do in order to explore as many parts of the system's interface as possible. In addition, *ad hoc* approaches to such study development may mean the testing is not easily repeatable on subsequent implementations or updates, and also that particular areas of the software may not be evaluated at all. In this paper we describe two (very different) approaches to designing an evaluation study for the same piece of software and discuss both the approaches taken, the differing results found and our comments on both of these.

Keywords: Usability studies, evaluation, UI Design, formal models

1 Introduction

There have been many investigations into the effectiveness of different types of usability testing and evaluation techniques, see for example (Nielsen & Landauer 1993) and (Doubleday et al. 1997) as well as research into the most effective ways of running the various types of studies (numbers of participants, expertise of testers, time and cost considerations *etc.*) (Nielsen 1994), (Lewis 2006). Our interest, however, is in a particular type of usability study, that of user evaluations. We are interested in how such studies are developed, *e.g.* what is the basis for the activities performed by the participants? In particular, given an implementation (or partial implementation) to test, is there a difference between the sort of study the developer of the system under test might produce and that of an impartial person, and if so do they produce different results? It is well known by the software engineering community that functional and behavioural testing is best performed by someone other than the software's developer. Often this can be achieved because there is a structured mechanism in place for devising tests, for example using model-based testing (Utting & Legeard 2006) or by having initial specifications that can be understood by experienced test developers (Bowen & Hinchey 1995), or at least by writing the tests before any code is written (as in test-driven or test-first development (Beck 2003)).

For user evaluation of software, and in particular the user interface of software, we do not have the sort of structured mechanisms for developing evaluation studies (or models upon which to base them) as we do for functional testing. Developing such studies relies on having a good enough knowledge of the software to devise user tasks that will effectively test the software, which for smaller scale development often means the software's developer. Given that we know it is not a good idea for functional testing to be carried out by the software's developer we suggest it may also be true that running, and more importantly, developing, user evaluations should not be done by the developer for the same reasons. This then presents the problem of how someone other than the software's developer can plan such a study without the (necessary) knowledge about the system they are testing.

In this paper we present an investigation into the differences between two evaluation studies developed using different approaches. The first was developed in the 'usual way' (which we discuss and define further in the body of the paper) by the developer of the software-under-test. The second was developed based on formal models of the software-under-test and its user interface (UI) by an independent practitioner with very little knowledge of the software-under-test prior to the modelling stage. We discuss the different outcomes of the two studies and share observations on differences and similarities between the studies and the results.

We begin by describing the software used as the basis for both evaluation studies. We then describe the process of deriving and running the first study along with the results. This is followed by a description of the basis and process of deriving the second study as well as the results of this. We then present a comparison of the two studies and their results, and finish with our conclusions.

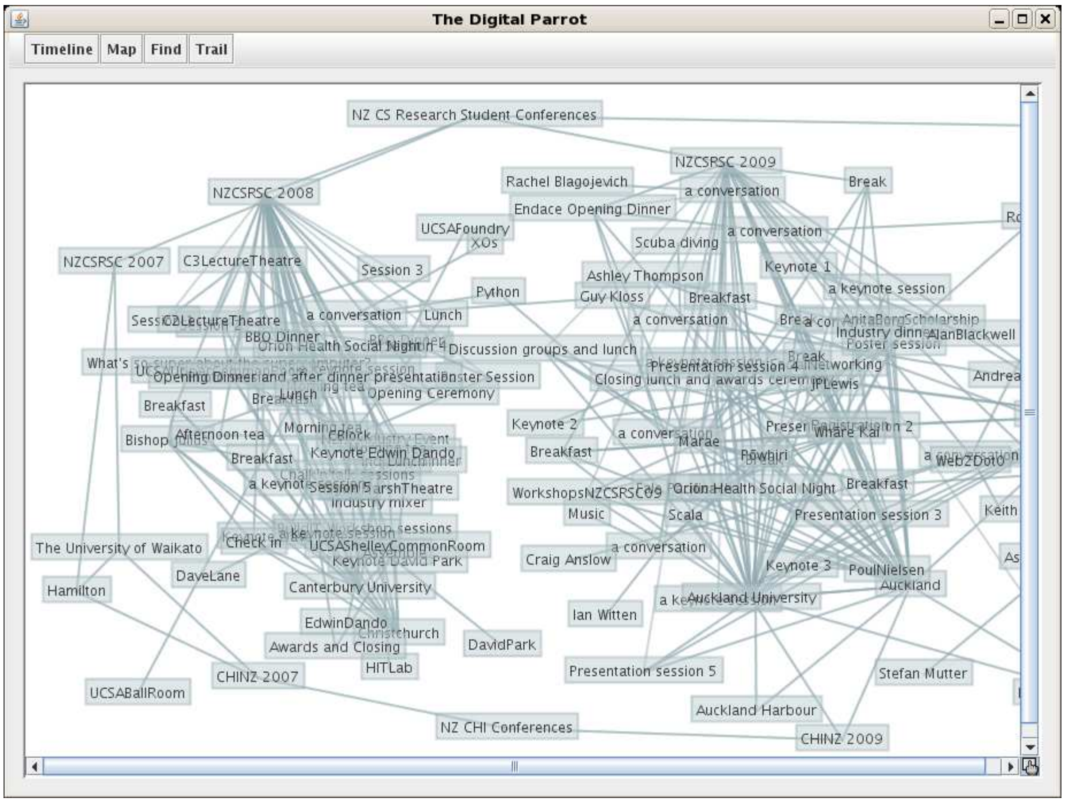
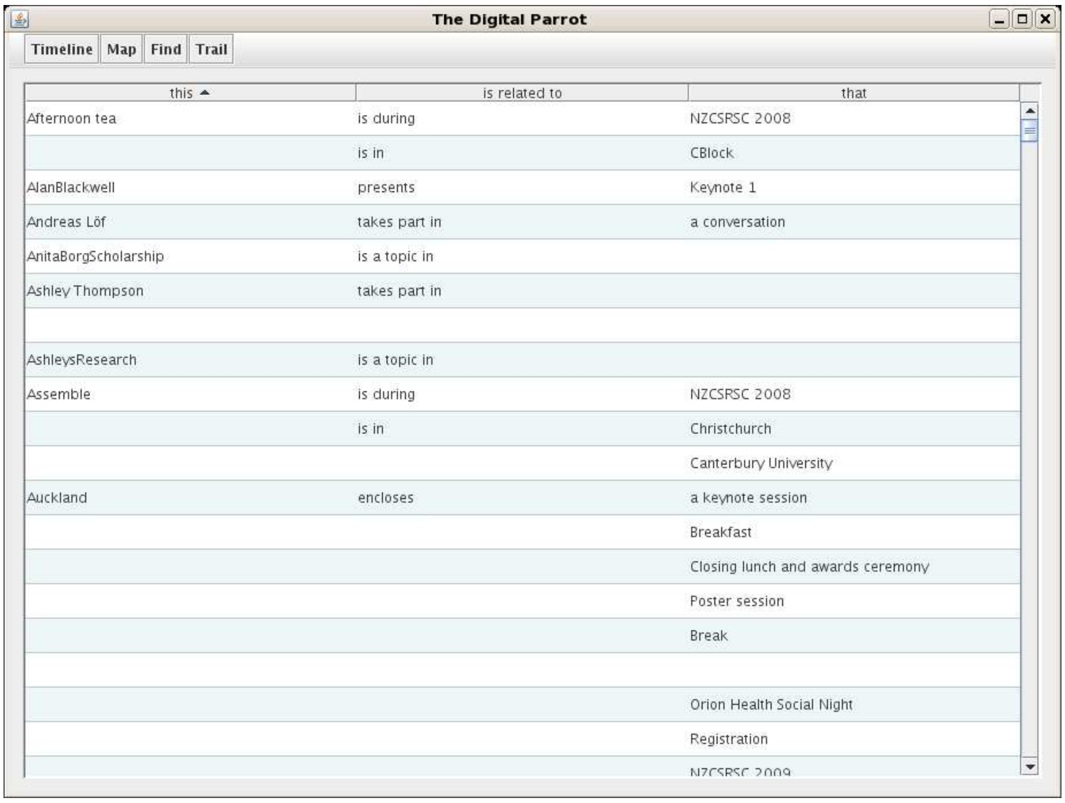


Figure 1: Graph Version of the Digital Parrot


The Digital Parrot

Timeline Map Find Trail

this ▲	is related to	that
Afternoon tea	is during	NZCSRSC 2008
	is in	CBlock
AlanBlackwell	presents	Keynote 1
Andreas Löf	takes part in	a conversation
AnitaBorgScholarship	is a topic in	
Ashley Thompson	takes part in	
AshleysResearch	is a topic in	
Assemble	is during	NZCSRSC 2008
	is in	Christchurch
		Canterbury University
Auckland	encloses	a keynote session
		Breakfast
		Closing lunch and awards ceremony
		Poster session
		Break
		Orion Health Social Night
		Registration
		NZCSRSC 2009

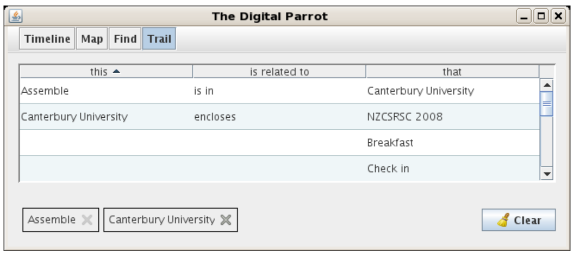
Figure 2: List Version of the Digital Parrot

2 The Digital Parrot Software

The Digital Parrot (Schweer & Hinze 2007), (Schweer et al. 2009) is a software system intended to augment its user's memory of events of their own life. It has been developed as a research prototype to study how people go about recalling memories from an augmented memory system.

The Digital Parrot is a repository of memories. Memories are encoded as subject–predicate–object triples and are displayed in the system's main view in one of two ways: a graph view and a list view, shown in Figures 1 and 2. Both views visualise the triple structure of the underlying memory information and let the user navigate along connections between memory items. The type of main view is chosen on program start-up and cannot be modified while the program is running.

The user interface includes four different navigators that can influence the information shown in the main view either by highlighting certain information items or by hiding certain information items. These navigators are the timeline navigator (for temporal navigation; shown in Figure 4), the map navigator (for navigation based on geospatial location), textual search and the trail navigator (for navigation based on information items' types and connections; shown in Figure 3).



this	is related to	that
Assemble	is in	Canterbury University
Canterbury University	encloses	NZCSRSC 2008
		Breakfast
		Check in

Figure 3: Trail Navigator

3 The First Study

At the time of the first study, the first development phase had ended and the Digital Parrot was feature-complete. Before using the software in a long-term user study (not described in this paper), we wanted to conduct a user evaluation of the software. Insights gained from the usability study would be used to form recommendations for changing parts of the Digital Parrot's user interface in a second development phase.

3.1 Goals

The first study had two main goals. The first was to detect any serious usability flaws in the Digital Parrot's user interface before using it in a long-term user study. We wanted to test how well the software could be used by novice users given minimal instructions. This mode of operation is not the typical mode for a system such as the Digital Parrot but was chosen to cut down on the time required by the participants as it removed the need to include a training period.

The second goal was to find out whether the participants would understand the visualisations and the purpose of the four different navigators.

3.2 Planning the Study

The study was run as a between-groups design, with half the participants assigned to each main view type (graph *vs* list view). We designed the study as a task-based study so that it would be easier to compare findings between participants. We chose a set of four tasks that we thought would cover all of the Digital Parrot's essential functionality. These tasks are as follows:

1. *To whom did [the researcher] talk about scuba diving? Write their name(s) into the space below.*
2. *Which conferences did [the researcher] attend in Auckland? Write the conference name(s) into the space below.*
3. *At which conference(s) did [the researcher] speak to someone about Python during the poster session? Write the conference name(s) into the space below.*
4. *In which place was the NZ CHI conference in 2007? Write the place name into the space below.*

The tasks were chosen in such a way that most participants would not be able to guess an answer. We chose tasks that were not too straightforward to solve; we expected that a combination of at least two of the Digital Parrot's navigators would have to be used for each task. Since the Digital Parrot is intended to help users remember events of their own lives, all tasks were phrased as questions about the researcher's experiences recorded in the system. The questions mimic questions that one may plausibly find oneself trying to answer about one's own past.

To cut down on time required by the participants, we split the participants into two groups of equal size. Each group's participants were exposed to only one of the two main view types. Tasks were the same for participants in both groups.

In addition to the tasks, the study included an established usability metric, the Systems Usability Scale (SUS) (Brooke 1996). We modified the questions according to the suggestions by Bangor *et al.* (Bangor et al. 2009). We further changed the questions by replacing "the system" with "the Digital Parrot". The intention behind including this metric was to get an indication of the severity of any discovered usability issues.

3.3 Participants and Procedure

The study had ten participants. All participants were members of the Computer Science Department at the University of Waikato; six were PhD students and four were members of academic staff. Two participants were female, eight were male. The ages ranged from 24 to 53 years (median 38, IQR 15 years). Participants were recruited via e-mails sent to departmental mailing lists and via personal contacts. Participants were not paid or otherwise rewarded for taking part in the usability test.

In keeping with University regulations on performing studies with human participants, ethical consent to run the study was applied for, and gained. Each participant in the study would receive a copy of their rights as a participant (including their right to withdraw from the study) and sign a consent form.

After the researcher obtained the participant's consent, they were provided with a workbook. The workbook gave a quick introduction to the purpose of the usability test and a brief overview of the system's features. Once the participant had read the first page, the researcher started the Digital Parrot and briefly demonstrated the four navigators (see Section 2). The participant was then asked to use the Digital Parrot to perform the four tasks stated in the workbook.

Each participant was asked to think aloud while using the system. The researcher took notes. After the tasks were completed, the participant was asked to fill in the SUS questionnaire about their experience with the Digital Parrot and to answer some questions about their background. The researcher would then ask a few questions to follow up on observations made while the participant was working on the tasks.

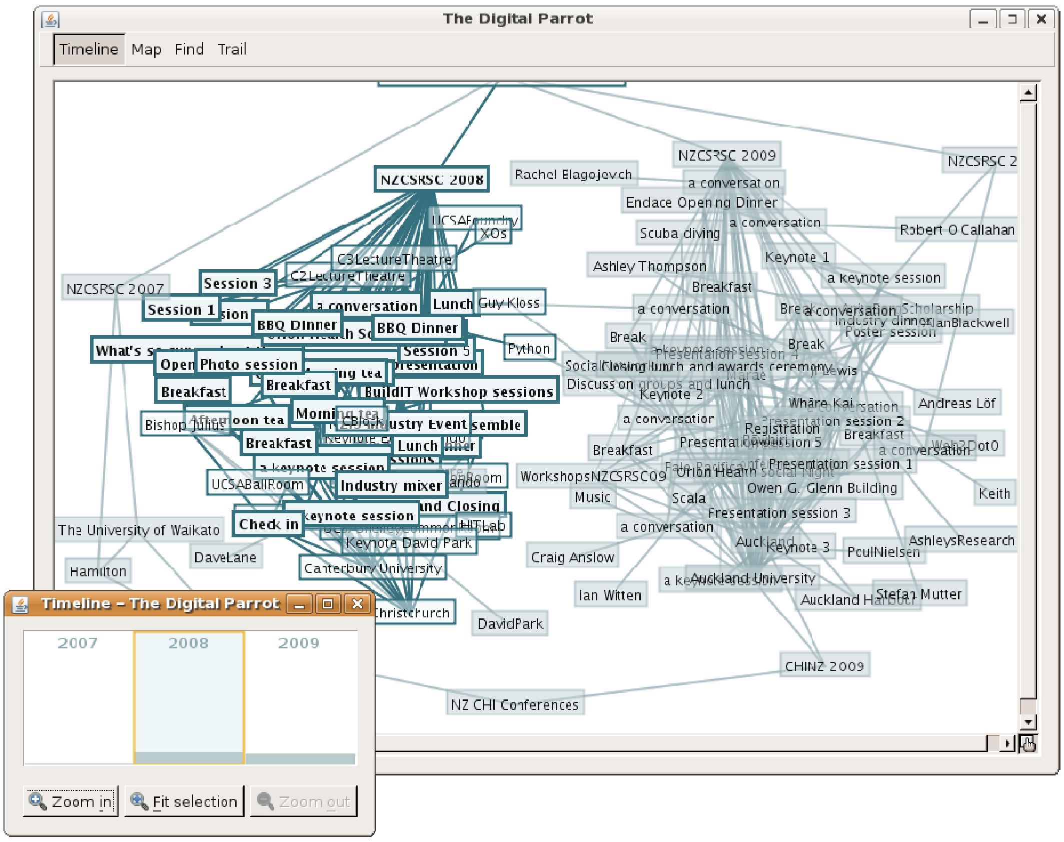


Figure 4: Timeline Navigator

3.4 Expectations

We did not have any particular expectations related to the first goal of this study, that of detecting potential usability problems within the Digital Parrot.

We did, however, have some expectations related to the second goal. The study was designed and conducted by the main researcher of the Digital Parrot project, who is also the main software developer. Thus, the study was designed and conducted by someone intimately familiar with the Digital Parrot's user interface, with all underlying concepts and also with the test data. For each of the tasks, we were aware of at least one way to solve the task with one or more of the Digital Parrot's navigators. We expected that the participants in the study would discover at least one of these ways and use it to complete the task successfully.

3.5 Results

The median SUS score of the Digital Parrot as determined in the usability test is 65 (min = 30, max = 92.5, IQR = 35), below the cut-off point for an acceptable SUS score (which is 70). The overall score of 65 corresponds to a rating between "ok" and "good" on Bangor *et al.*'s adjective scale (Bangor et al. 2009). The median SUS score in the graph condition alone is 80 (min = 42.5, max = 92.5, IQR = 40), which indicates an acceptable user experience and corresponds to a rating between "good" and "excellent" on the adjective scale. The median SUS score in the list condition is 57.5 (min = 30, max = 77.5, IQR = 42.5). The difference in SUS score is not statistically significant but does reflect our observations that the users in the list condition found the system harder to use than those in the graph condition.

Based on our observations of the participants in this study, we identified nine major issues with the Digital Parrot's user interface as well as several smaller problems. In terms of our goals for the study we found that there were usability issues, some we would consider major, which would need to be addressed before conducting a long-term user study. In addition there were some issues with the use of navigators. The details of our findings are as follows:

1. Graph: The initial view is too cluttered.
2. Graph: The nature of relationships is invisible.
3. List: The statement structure does not become clear.
4. List: Text search does not appear to do anything.
5. Having navigators in separate windows is confusing.
6. The map is not very useful.
7. Users miss a list of search results.
8. The search options could be improved.
9. The trail navigator is too hard to use.

Some of these issues had obvious solutions based on our observations during the study and on participants' comments. Other issues, however, were less easily resolved.

We formulated seven recommendations to change the Digital Parrot's user interface:

1. Improve the trail navigator's user interface.
2. Improve the map navigator and switch to a different map provider.
3. De-clutter the initial graph view.
4. Enable edge labels on demand.

5. Highlight statement structure more strongly in list.
6. Change window options for navigators.
7. Improve the search navigator.

As can be seen, these recommendations vary greatly in scope; some directly propose a solution while others require further work to be done to find a solution.

4 The Second Study

4.1 Goals

The goals of our second study were to emulate the intentions of the first, that is try to find any usability or functional problems with the current version of the Digital Parrot software. In addition, the intention was to develop the study with no prior knowledge of the software or of the first study (up to the point where we could no longer proceed without some of this information) by using abstract tests derived from formal models of the software and its user interface (UI) as the basis for planning the tasks of the study. We were interested to discover if such abstract tests could be used to derive an evaluation study in this way, and if so how would the results differ from those of the initial study (if at all).

4.2 Planning the Study

The first step was to obtain a copy of the Digital Parrot software and reverse-engineer it into UI models and a system specification. In general, our assumption is that such models are developed during the design phase and prior to implementation rather than by reverse-engineering existing systems. That is, a user-centred design (UCD) approach is taken to plan and develop the UI prior to implementation and these are used as the basis for the models. For the purposes of this work, however, the software had been implemented already and as such a specification and designs did not exist, hence the requirement to perform reverse-engineering. This was done manually, although tools for reverse-engineering software in this manner do exist (for example GUI Ripper[9]), but not for the models we planned to use.

We spent several days interacting with the software and examining screen shots in order to begin to understand how it worked. We wanted to produce as much of the model as possible before consulting with the software designer to 'fill in the gaps' where our understanding was incomplete. Of course, such a detailed examination of the software was itself a form of evaluation, as by interacting with the software comprehensively enough to gain the understanding required for modelling, we formed our own opinions about how usable, or how complex, parts of the system were. However, in general where models and tests are derived prior to implementation this would not be the case. The first study had already taken place but the only information we required about this study was the number and *type* of participants used (so that we could use participants from the same demographic group) and the fact that the study was conducted as a between-groups study with both graph and list versions of the software being tested. We had no preconceived ideas of how our study might be structured at this point, the idea being that once the models were completed and the abstract tests derived we would try and find some structured way of using these to guide the development of our study.

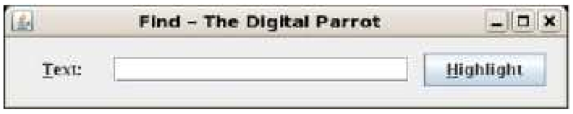


Figure 5: Find Dialogue

4.3 The Models

We began the modelling process by examining screenshots of the Digital Parrot. This enabled us to identify the widgets used in the various windows and dialogues of the system that provided the outline for the first set of models. We used presentation models and presentation interaction models (PIMs) from the work described in (Bowen & Reeves 2008) as they provide a way of formally describing UI designs and UIs with a defined process for generating abstract tests from the models (Bowen & Reeves 2009). Presentation models describe each dialogue or window of a software system in terms of its component widgets, and each widget is described as a tuple consisting of a name, a widget category and a set of the behaviours exhibited by that widget. Behaviours are separated into S-behaviours, which relate to system functionality (*i.e.* behaviours that change the state of the underlying system) and I-behaviours that relate to interface functionality (*i.e.* behaviours relating to navigation or appearance of the UI).

Once we had discovered the structure of the UI and created the initial model we then spent time using the software and discovering what each of the identified widgets did in order to identify the behaviours to add to the model. For some parts of the system this was relatively easy, but occasionally we were unable to determine the behaviour by interaction alone. For example, the screenshot in figure 5 shows the "Find" dialogue from the Digital Parrot, from which we developed the following presentation model:

```
FindWindow is
  (SStringEntry, Entry, ())
  (HighlightButton, ActionControl,
                          (S_HighlightItem))
  (FMinIcon, ActionControl, (I_FMinToggle))
  (FMaxIcon, ActionControl, (I_FMaxToggle))
  (FXIcon, ActionControl, (I_Main))
  (HSCKey, ActionControl, (S_HighlightItem))
  (TSCKey, ActionControl, (?))
```

We were unable to determine what the behaviour of the shortcut key option *Alt-T* was and so marked the model with a "?" as a placeholder. Once the presentation models were complete we moved on to the second set of models, the PIMs, which describe the navigation of the interface. Each presentation model is represented by a state in the PIM and transitions between states are labelled with I-behaviours (the UI navigational behaviours) from those presentation models. PIMs are described using the μCharts language (Reeve 2005), which enables each part of the system to be modelled within a single, sequential μchart that can then be composed together or embedded in states of other models to build the complete model of the entire system. Figure 6 shows one of the PIMs representing part of the navigation of the "Find" dialogue and "Main" window.

In the simplest case, a system with five different windows would be described by a PIM with five states (each state representing the presentation model for one of the windows). However, this assumes that each of the windows is modal and does not interact with

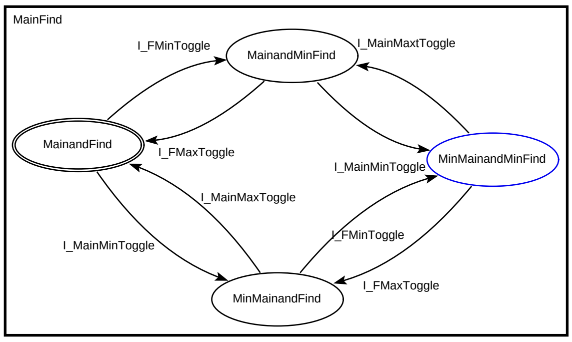


Figure 6: PIM for Main and Find Navigation

any of the other windows. In the Digital Parrot system none of the dialogues are modal, in addition each of the windows can be minimised but continues to interact with other parts of the system while in its minimised state. This led to a complex PIM consisting of over 100 states. The complexity of the model and of the modelling process (which at times proved both challenging and confusing) gave us some indication of how users of the system might be similarly confused when interacting with the system in its many various states. Even before we derived the abstract tests, therefore, we began to consider areas of the system we would wish to include in our evaluation study (namely how the different windows interact).

The third stage of the modelling was to produce a formal specification of the functionality of the Digital Parrot. This was done using the Z specification language (ISO 2002) and again we completed as much of the specification as was possible but left some areas incomplete where we were not confident we completely understood all of the system's behaviour. The Z specification consists of a description of the state of the system (which describes the data for the memory items stored in the system as sets of observations on that data) and operations that change that state. For example the "SelectItems" operation is described in the specification as:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{SelectItems}} \\ \hline
\Delta \mathit{DPSystem} \\
i? : \mathit{Item} \\
li? : \mathit{Item} \\ \hline
\mathit{AllItems}' = \mathit{AllItems} \\
\mathit{SelectedItems}' = \mathit{SelectedItems} \cup \{li?\} \cup \{i?\} \\
li?.\mathit{itemName} = (i?.\mathit{itemLink}) \\
\mathit{VisibleItems}' = \mathit{VisibleItems} \\
\mathit{HiddenItems}' = \mathit{HiddenItems} \\
\mathit{TrailItems}' = \mathit{TrailItems} \\ \hline
\end{array}$$

The meaning of this is that the operation consists of observations on the *DPSystem* state before and after the operation takes place (denoted by $\Delta \mathit{DPSystem}$) and there are two inputs to the operation *li?* and *i?*, which are both of type *Item*. After the operation has occurred some observations are unchanged. Observations marked with $'$ indicate they are after the operation, so, for example $\mathit{AllItems}' = \mathit{AllItems}$ indicates this observation has not changed as a result of the operation. The *SelectedItems* observation does change however, and after the operation this set is increased to include the inputs *li?* and *i?*, which represent the new items selected.

Once we had completed as much of the modelling as was possible we met with the software's developer to firstly ensure that the behaviour we had described was correct, and secondly to fill in the gaps in the areas where the models were incomplete. With a complete set of models and a complete specification we were then able to relate the UI behaviour to the specified functionality by creating a relation between the S-behaviours of the presentation models (which relate to functionality of the system) and the operations of the specification. This gives a formal description of the S-behaviours by showing which operations of the specification they relate to, and the specification then gives the meaning. Similarly, the meaning of the I-behaviours is given by the PIM. The relation, which we call the presentation model relation (PMR), we derived is shown below:

$$\begin{array}{l}
\mathit{S\_HighlightItems} \mapsto \mathit{SelectItems} \\
\mathit{S\_PointerMode} \mapsto \mathit{TogglePointerMode} \\
\mathit{S\_CurrentTrail} \mapsto \mathit{SelectCurrentTrailItems} \\
\mathit{S\_SelectItemMenu} \mapsto \mathit{MenuChoice} \\
\mathit{S\_ZoomInTL} \mapsto \mathit{TimelineZoomInSubset} \\
\mathit{S\_ZoomOutTL} \mapsto \mathit{TimelineZoomOutSuperset} \\
\mathit{S\_SelectItemsByTime} \mapsto \mathit{SelectItemsByTime} \\
\mathit{S\_FitSelectionByTime} \mapsto \mathit{FitItemsByTime} \\
\mathit{S\_HighlightItem} \mapsto \mathit{SelectByName} \\
\mathit{S\_AddToTrail} \mapsto \mathit{AddToTrail} \\
\mathit{S\_ZoomInMap} \mapsto \mathit{RestrictByLocation} \\
\mathit{S\_ZoomOutMap} \mapsto \mathit{RestrictByLocation} \\
\mathit{S\_Histogram} \mapsto \mathit{UpdateHistogram}
\end{array}$$

This completed the modelling stage and we were now ready to move on to derivation of the abstract tests that we describe next.

4.4 The Abstract Tests

Abstract tests are based on the conditions that are required to hold in order to bring about the behaviour given in the models. The tool which we use for creating the presentation models and PIMs, called PIMed (PIMed 2009) has the ability to automatically generate a set of abstract tests from the models, but for this work we derived them manually using the process described in (Bowen & Reeves 2009). Tests are given in first-order logic. The informal, intended meaning of the predicates can initially be deduced from their names, and are subsequently formalised when the tests are instantiated. For example, two of the tests that were derived from the presentation model and PIM of the "Find" dialogue and "MainandFind" are:

$$\begin{array}{l}
\mathit{State}(\mathit{MainFind}) \Rightarrow \\
\mathit{Visible}(\mathit{FXIcon}) \wedge \mathit{Active}(\mathit{FXIcon}) \wedge \\
\mathit{hasBehaviour}(\mathit{FXIcon}, \mathit{I\_Main})
\end{array}$$

$$\begin{array}{l}
\mathit{State}(\mathit{MainFind}) \Rightarrow \\
\mathit{Visible}(\mathit{HighlightButton}) \wedge \\
\mathit{Active}(\mathit{HighlightButton}) \wedge \\
\mathit{hasBehaviour}(\mathit{HighlightButton}, \mathit{S\_HighlightItem})
\end{array}$$

The first defines the condition that when the system is in the *MainFind* state a widget called *FXIcon* should be visible and available for interaction (active) and when interacted with should generate the interaction behaviour called *I_Main*, whilst the second requires that in the same state a widget called *HighlightButton* is similarly visible and available for interaction and generates the system behaviour *S_HighlightItem*. When we come to instantiate the test we use the PIM to determine the meaning of the I-behaviours and the Z specification (via the PMR) to determine the meaning of the S-behaviours. The set of tests also includes conditions describing what it means for the UI to be in any named state so that this can similarly be tested. The full details of this are given in (Bowen & Reeves 2009) and are beyond the scope of this paper. Similar tests are described for each of the widgets of the models, *i.e.* for every widget in the model there is at least one corresponding test. The abstract tests consider both the required functionality within the UI (S-behaviour tests) as well as the navigational possibilities (I-behaviour tests). As there are two different versions of the Digital Parrot software, one that has a graph view for the data and the other a list view, there were two slightly different sets of models. However, there was very little difference between the two models (as both versions of the software have almost identical functionality); there were in fact three behaviours found only in the list version of the software (all of which relate to the UI rather than underlying functionality) and one UI behaviour found only in the graph version. This gave rise to four abstract tests that were unique to the respective versions.

4.5 Deriving The Study

With the modelling complete and a full set of abstract tests, we now began to consider how we could use these to derive an evaluation study. To structure the study we first determined that all S-behaviours should be tested by way of a specific task (to ensure that the functional behaviour could be accessed successfully by users). The relation we showed at the end of section 4.3 lists thirteen separate S-behaviours. For each of these we created an outline of a user task, for example from the test:

$$\begin{array}{l} State(MainFind) \Rightarrow \\ Visible(HighlightButton) \\ \wedge\ Active(HighlightButton) \wedge \\ hasBehaviour(HighlightButton, S\_HighlightItem) \end{array}$$

we decided that there would be a user task in the study that would require interaction with the "Find" dialogue to utilise the *S_HighlightItem* behaviour. To determine what this behaviour is we refer to the relation between specification and S-behaviours, so in this case we are interested in the Z operation *SelectByName*. We generalised this to the task:

> *Use the "Find" dialogue to highlight a specified item.*

Once we had defined all of our tasks we made these more specific by specifying actual data items. In order to try and replicate conditions of the first study we used (where possible) the same examples. The "Find" task, for example, became the following in our study:

> *Use the "Find" functionality to discover which items are connected to the item called "Scuba Diving"*

In order to complete this task the user will interact with the "Find" dialogue and use the *S_HighlightItem* behaviour that relates to the functionality in the system, *SelectByName*, which subsequently highlights the item given in the "Find" text field as well as all items connected to it.

Once tasks had been defined for all of the S-behaviours we turned our attention to the I-behaviours. It would not be possible to check *all* navigational possibilities due to the size of the state space for the models of this system (and this may be true for many systems under test) as this would make the study too long. What we aimed to do instead was maximize the coverage of the PIM by following as many of the I-behaviour transitions as possible and so we ordered the S-behaviour tasks in such a way as to maximise this navigation. Having the PIM as a visualisation and formalisation of the navigational possibilities enables us to clearly identify exactly how much we are testing. We also wanted to ensure that the areas of concern we had identified as we built the models were tested by the users. As such, we added tasks that relied on the interaction of functions from multiple windows. For example we had a task in our study:

> *What are the items linked to the trail "NZ CS Research Student Conferences" which took place in the North Island of New Zealand?*

that required interaction with the *Trail* window and the *Map* window at the same time.

Our final study consisted of eighteen tasks for the graph version of the software and seventeen for the list version (the additional task relating to one of the functions specific to the graph version). After each task we asked the user to rate the ease with with they were able to complete the task and provided the opportunity for them to give any comments they had about the task if they wished. Finally we created a post-task questionnaire for the participants that was aimed at recording their subjective feelings about the tasks, software and study.

4.6 Participants and Procedure

As with the first study, ethical consent to run the second study was sought, and given. We recruited our participants from the same target group, and in the same way, as for the initial study with the added criterion that no one who had participated in the initial study would be eligible to participate in the second.

We used a "between groups" methodology with the ten participants being randomly assigned either the Graph view version of the software or the List view. The study was run as an observational study with notes being taken of how participants completed, or attempted to complete, each task. We also recorded how easily *we* perceived they completed the tasks and subsequently compared this with the participants' perceptions. Upon completion of the tasks the participants were asked to complete the questionnaire and provide any other feedback they had regarding any aspect of the study. Each study took, on average, an hour to complete.

4.7 Results

The second study found five functionality bugs and twenty seven usability issues. The bugs found were:

1. Continued use of "Zoom in" on map beyond maximum ability causes a graphics error.
2. The shortcut keys 'i' and 'o' for "Zoom in" and "Zoom out" on Map don't work.
3. Timeline loses capability to display 2009 data once it has been interacted with.
4. Data items visualised in timeline move around between months.

5. Labels on the Map sometimes move around and sit on top of each other during zoom and move operations.

We had identified two of the functionality bugs during the modelling stage (the loss of 2009 data and the non-functional shortcut keys on the map) and in a traditional development/testing scenario we would expect that we would report (and fix) these prior to the evaluation. The usability issues ranged from minor items to major issues. An example of a minor issue was identifying that the widget used to toggle the mouse mode in the graph view is very small and in an unusual location (away from all other widgets) and is easily overlooked. We consider this minor because once a user knows the widget is there it is no longer a problem. An example of a major issue was the lack of feedback from the "Find" function that meant it was impossible to tell whether or not it had returned any results in many cases.

We made 27 recommendations for changes to the software. Some of these were specific changes that could be made to directly fix usability issues found, such as the recommendation:

> "Inform users if "Find" returns no results."

Whereas others were more general and would require further consideration, for example the recommendation:

> "Reconsider the use of separate dialogues for features to reduce navigational and cognitive workload."

Other observations led us to comment on particular areas of the software and interface without directly making recommendations on how these might be addressed. For example, we observed that most participants had difficulty understanding conceptually how "Trails" worked and what their meaning was. This led to difficulties with tasks involving "Trails" but also meant that participants used "Trails" when not required to do so as they found it had a particular effect on the data that they did not fully understand, but that enabled them to visualise information more easily. This is related to the fact that the amount of data and the way it is displayed and the use of highlighting were all problematic for some of the tasks.

We also determined from our comparison of the measure of ease with which tasks were completed against the participants' perception that for many participants even when they successfully completed a task they were not confident that they had done so. For example we would record that a participant completed a task fairly easily as they took the minimum number of steps required and were successful, but the participant would record that they completed the task with difficulty. This was also evident from the behaviour of participants as they would often double check their result to be certain it was correct. This is linked to the overall lack of feedback and lack of visibility that was reported as part of our findings.

From the results of the second study we were confident that the study produced from the formal models was able to find both specific problems (such as functionality bugs) and usability problems, as well as identify general overall problems such as lack of user confidence.

5 Comparing The Two Studies

There was an obvious difference in the tasks of the two studies. In the first study users were given four tasks requiring several steps and were able to try and achieve this in any way they saw fit, whereas in the second study they were given twenty seven tasks which were defined very specifically. This meant that the way in which participants interacted with the software whilst carrying out these tasks was very different. In the first study users were encouraged to interact with the software in the way that seemed most natural to them in order to complete the tasks, whereas in the second study they were given much clearer constraints on *how* they should carry out a particular task (for example: "Use the "Find" function to....") to ensure they interacted with specific parts of the system. While this meant that coverage of functionality and navigation was more tightly controlled in the second study (which is beneficial in ensuring as many problems and issues as possible are found) it also meant that it did not provide a clear picture of how users would actually interact with the software outside of the study environment and as such led to the reporting of problems that were in fact non-issues (such as the difficulties users had in interpreting the amount of data for a time period from the histogram).

One of the problems with the tasks of the initial study, however, was that by allowing users to interact in any way they chose particular parts of the system were hardly interacted with at all, which meant that several issues relating to the "Timeline" that were discovered during the second study were not evident in the first study due to the lack of interaction with this functionality.

The other effect of the way the tasks were structured was the subjective satisfaction measurements of the participants. The participants of the first study were more positive about their experience using the software than those of the second study. We feel that this is partly due their interactions and the fact that the first group had a better understanding of the software and how it might be used in a real setting than the second group did. However, there is also the possibility that the participants of the first study moderated their opinions because they knew that the researcher conducting the study was also the developer of the software (which is one of the concerns we were hoping to address with our work).

6 Reflections on Process and Outcomes

One of the things we have achieved by this experiment is an understanding of how formal models might be used to develop a framework for developing user evaluations. This work shows that a study produced in such a way is as good (and in some cases better) at discovering both usability and functional problems with software. It is also clear, however, that the type of study produced does not allow for analysis of utility and learnability from the perspective of a user encouraged to interact as they choose with software.

Some of the advantages of this approach are: the ability to clearly identify the scope of the study with respect to the navigational possibilities of the software-under-test (via the PIM); a framework to identify relevant user tasks (via the abstract tests); a mechanism to support creation of oracles for inputs/outputs to tasks (via the specification). This supports our initial goal of supporting development of evaluation studies by someone other than the software developer as it provides structured information to support this. However, it also leads to an artificial approach to interacting with the software and does not take into account the ability of participants to learn through exploration and as such may discover usability issues which are unlikely to occur in a real-world use of the software as well as decrease subjective satisfaction of participants with the software.

7 Conclusion

It seems clear that there is no 'one size fits all' approach to developing evaluation studies, as the underlying goals and intentions must play a part in how the tasks are structured. However, it does appear that the use of formal models in the ways shown here can provide a structure for determining what those tasks should be and suggests ways of organising them to maximise interaction. Perhaps using both methods (traditional and formally based) is the best way forward. Certainly there are benefits to be found from taking the formal approach, and for developers with no expertise in developing evaluation studies this process may prove supportive and help them by providing a framework to work within. Similarly for formal practitioners who might otherwise consider usability testing and evaluation as too informal to be useful the formal structure might persuade them to reconsider and include this important step within their work. The benefits of a more traditional approach are the ability to tailor the study for discovery as well as evaluation, something the formally devised study in its current form was not good at at all. Blending the two would be a valuable way forward so that we can use the formal models as a framework to devise structured and repeatable evaluations, and then extend or develop the study with a more human-centred approach that allows for the other benefits of evaluation that would otherwise be lost.

8 Future Work

We would like to take the joint approach described to develop a larger study. This would enable us to see how effective combining the methods might be, how well the approach scales up to larger software systems and studies, and where the difficulties lie in working in this manner. We have also been looking at reverse-engineering techniques and tools which could assist when working with existing or legacy systems and this work is ongoing.

9 Acknowledgments

Thanks to all participants of both studies.

References

Bangor, A., Kortum, P. & Miller, J. (2009), 'Determining what individual SUS scores mean: Adding an adjective rating scale', *Journal of Usability Studies* **4**(3), 114–123.

Beck, K. (2003), *Test-Driven Development: By Example*, The Addison-Wesley Signature Series, Addison-Wesley.

Bowen, J. & Reeves, S. (2008), 'Formal models for user interface design artefacts', *Innovations in Systems and Software Engineering* **4**(2), 125–141.

Bowen, J. & Reeves, S. (2009), Ui-design driven model-based testing, *in* 'M. Harrison and M. Massink (eds.) Proceedings of 3rd International Workshop on Formal Methods for Interactive Systems (FMIS'09)', Electronic Communications of the EASST, 22.

Bowen, J. P. & Hinchey, M. G., eds (1995), *Improving Software Tests Using Z Specifications*, Vol. 967 of *Lecture Notes in Computer Science*, Springer.

Brooke, J. (1996), SUS: A "quick and dirty" usability scale, *in* P. W. Jordan, B. Thomas, I. L. McClelland & B. A. Weerdmeester, eds, 'Usability evaluation in industry', CRC Press, chapter 21, pp. 189–194.

Doubleday, A., Ryan, M., Springett, M. & Sutcliffe, A. (1997), A comparison of usability techniques for evaluating design, *in* 'DIS '97: Proceedings of the 2nd conference on Designing interactive systems', ACM, New York, NY, USA, pp. 101–110.

ISO (2002), *ISO/IEC 13568— Information Technology—Z Formal Specification Notation—Syntax, Type System and Semantics*, Prentice-Hall International series in computer science, first edn, ISO/IEC.

Lewis, J. R. (2006), 'Sample sizes for usability tests: mostly math, not magic', *interactions* **13**(6), 29–33.

Nielsen, J. (1994), *Usability Engineering*, Morgan Kaufmann Publishers, San Francisco, California.

Nielsen, J. & Landauer, T. K. (1993), A mathematical model of the finding of usability problems, *in* 'CHI '93: Proceedings of the INTERACT '93 and CHI '93 conference on Human factors in computing systems', ACM, New York, NY, USA, pp. 206–213.

PIMed (2009). PIMed : An editor for presentation models and presentation interaction models, `http://sourceforge.net/projects/pims1/?source=directory`.

Reeve, G. (2005), A Refinement Theory for μCharts, PhD thesis, The University of Waikato.

Schweer, A. & Hinze, A. (2007), The Digital Parrot: Combining context-awareness and semantics to augment memory, *in* 'Proceedings of the Workshop on Supporting Human Memory with Interactive Systems (MeMos 2007) at the 2007 British HCI International Conference'.

Schweer, A., Hinze, A. & Jones, S. (2009), Trails of experiences: Navigating personal memories, *in* 'CHINZ '09: Proceedings of the 10th International Conference NZ Chapter of the ACM's Special Interest Group on Human-Computer Interaction', ACM, New York, NY, USA, pp. 105–106.

Utting, M. & Legeard, B. (2006), *Practical Model-Based Testing - A tools approach*, Morgan and Kaufmann.