FALSIFICATION TESTING FOR USABILITY INSPECTION METHOD ASSESSMENT

Alan Woolrych **Gilbert Cockton** **Mark Hindmarch**
School of Computing and Technology, University of Sunderland,
Sir Tom Cowie Campus, St Peters Way, Sunderland, SR6 0DD
alan.woolrych@sunderland.ac.uk gilbert.cockton@sunderland.ac.uk mark.hindmarch@sunderland.ac.uk

ABSTRACT

We need more reliable usability inspection methods (UIMs), but assessment of UIMs has been unreliable [5]. We can only reliably improve UIMs if we have more reliable assessment.

When assessing UIMs, we need to code analysts' predictions as true or false positives or negatives, or as genuinely missed problems. Defenders of UIMs often claim that false positives cannot be accurately coded, i.e., that a prediction is true but has never shown up through user testing or other validation approaches. We show this and similar claims to be mistaken by briefly reviewing methods for reliable coding of each of five types of prediction outcome. We focus on falsification testing, which allows confident coding of false positives.

Keywords

User testing, falsification testing, usability inspection methods, model-based methods.

1. INTRODUCTION

Currently, there are two general ways to *evaluate* a product's usability: predictions by analysts or user testing (with real participants). While the latter can be more reliable, achieving this is very labour intensive, time consuming and very expensive. When getting the product right first time is essential, there is a pressing need for more efficient methods of achieving satisfactory usability.

Usability Inspection Methods (UIMs) would appear to be a suitable approach here, as they are generally quicker, and consequently less expensive. However, speed and reduced cost come at a price since UIMs can be significantly less reliable than user testing. We thus need to improve UIM usage, but before we can, we must have reliable methods to assess UIMs. One key issue with UIM assessment is confident coding of analyst predictions from usability inspections. It is not a clear cut matter of successful predictions vs. missed problems. A major concern is the issue of false positives. How can we be sure that an analyst prediction is really a false positive? In fact, some believe that false positives don't exist, i.e., research simply fails to flush out predicted problems that really do exist!

2. HOW DO WE ASSESS UIMS?

UIMs are commonly assessed by their validity, thoroughness and effectiveness [9], even though percentages fail to comprehensively assess UIMs [10]. *Validity* drops as the number of problems found with a UIM exceeds the real problems found. Analysts make false predictions (false positives), as well as successful ones. Fewer false positives mean a more valid UIM.

$$\text{Validity} = \frac{\text{Count of real problems found using UIM}}{\text{Count of problems predicted by UIM}}$$

Note that if false positives do not exist, both counts must become identical and all UIMs have a perfect validity of 1! The *thoroughness* of a UIM rises as more of the real problems that exist that are found.

$$\text{Thoroughness} = \frac{\text{Count of real problems found using UIM}}{\text{Count of known usability problems}}$$

The *effectiveness* of a UIM is the (weighted) product of its thoroughness and validity [6]. To calculate this accurately, we must correctly code *all* analyst predictions. This is not just to get the right percentages. To understand how and why false positives and genuine misses arise, we must first be able to properly code analyst predictions.

3. ASSESSMENT VIA USER TESTING

The "count of known usability problems" used to calculate thoroughness can become known from:

- Helpdesk logs
- Logging (via software)
- Observation in real use
- User interviews
- User diaries
- User testing

User testing is thus just one approach. The approach below uses examples for user testing to validate UIMs, but much generalises to other empirical problem sources.

4. RELIABILITY IN VALIDATION VIA USER TESTS

Reliable assessment of UIMs (or model based methods) starts with accurate coding of predictions. We need to be absolutely sure that when we assess UIMs that a 'hit' really was a true positive and a 'miss' is a missed problem (i.e., not a false negative but a *genuine miss*). Accurate coding requires confidence in genuine misses and in classifying analysts' predictions as true or false positives or negatives.

Correct Coding	Result (miscoded as)	Source of Error
Genuine Miss *NOT DISCOVERED* *REAL PROBLEM* *(found in tests)*	True +ve	Incorrect extraction
True +ve (hit) *PREDICTED (discovered and confirmed)* *REAL PROBLEM*	False +ve	Incorrect extraction
		Incomplete coverage
	Genuine miss	Merging error
False +ve *PREDICTED* *NOT FOUND IN TESTS*	True +ve	Incorrect extraction
False –ve *NOT PREDICTED* *(discovered and eliminated)* *REAL PROBLEM*	True -ve	Incorrect extraction
		Incomplete coverage
	Genuine Miss	No evidence of elimination
True –ve *NOT PREDICTED* *NOT FOUND IN TESTS*	False -ve	Incorrect extraction
	Not detected	No evidence of elimination

Table 1: Sources of coding errors and their results

There are five types of (non)-prediction when assessing UIMs. Each has associated risks, as shown in Table 1. We can see that a problem that should be a *genuine miss* (correct coding) could be miscoded (result = true positive) if a problem is incorrectly extracted from empirical data (source of error, [7]) and matched to an analyst prediction. The same source of error miscodes a *false positive* as a hit.

A *true positive* (i.e., a 'hit') could be miscoded as a false positive from two sources of error: a real problem incorrectly extracted from empirical data; empirical coverage that misses the problem. A third source of error, poor merging of analyst predictions, results in miscoding as a genuine miss if a correct prediction was not kept separate from others — analyst agreement on problem identification can widely vary [7] and great care needs to be taken with merging and matching analyst predictions [2].

True and false negatives do not arise with all UIM usage, since instruments such as extended report formats are required to reveal analyst elimination of (im)probable problems [4]. Without evidence of elimination (source of error), a *false negative* will be miscoded as a genuine miss. Another source of error,

incorrect extraction from user test data, will result in miscoding as a hit (true positive).

For *true negatives*, incorrect extraction results in miscoding as false negatives. Without evidence of elimination, true negatives are undetectable, and thus analysts' abilities to spot fair to excellent design cannot be assessed.

There are thus four sources of error when validating UIMs:

1. *Incorrect extraction:* finding and describing problems from empirical data
2. *Incomplete coverage:* finding real problems (e.g., via user testing)
3. *Merging error:* incorrect or misleading merging of analyst predictions
4. *No evidence of elimination:* recording analyst decision making

Our approach to these errors has been developed over four years and applied in two major studies [2, 3]. The first source of error can be addressed by structured problem extraction methods such as SUPEX [1]. The last two sources of error can be addressed by Extended Structured Problem Report Formats (ESPRFs [4]). Within our DARe model for analyst behaviour [11], usability inspection has two distinct phases. Analysts discover possible problems, then analyse them (using a variety of knowledge resources in both discovery and analysis), keeping probable problems, and eliminating improbable ones. ESPRFs allow for the reporting of both false negatives and true negatives. Without ESPRFs, both true and false negatives will be miscoded unless the UIM explicitly records success and failure cases (e.g., [8]). We have not yet fully reported our approach to the second source of error, and we are often challenged to defend the existence of false negatives.

The first source of error is the most cited reason for mistrusting unfavourable UIM assessments. We have addressed this to ensure reliable validity scores over five years via *falsification testing*. As many still believe that false positives cannot be reliably coded, we explain our approach in detail to show that doubts are poorly grounded.

5. PERFORMING FALSIFICATION TESTING

The method for falsification testing involves the rigorous testing of analyst predictions via user testing (Figure 1). Analyst predictions are analysed and merged into a master problem set, taking care not to distort predictions. The key to falsification testing lies in analysing individual problems to identify likely user difficulties that should arise in testing. Within the context of the test application, task sets are systematically derived to expose these likely difficulties, that is, if the prediction is valid. Put simply, the individual predicted problems are 'stressed' via user testing to ensure a high level of confidence in final coding.

The principle is simple, if a prediction is accurate, then it will be confirmed by user testing. If a prediction does not materialise as a problem, we can have confidence that it does not exist, and that the particular prediction can be confidently coded as a false positive. Falsification testing ensures that false positive coding of predictions is not a consequence of incomplete coverage in user testing.

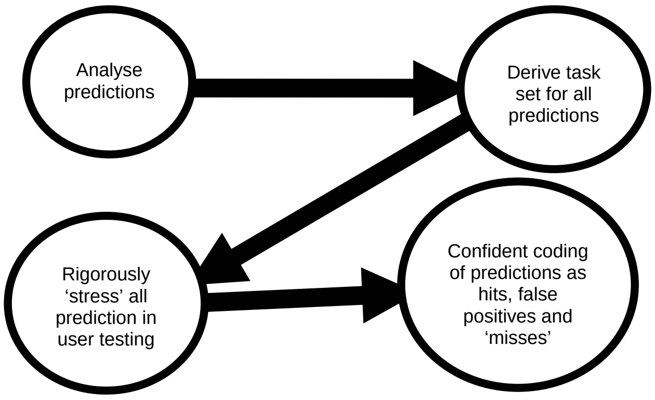
Analyse predictions
Derive task set for all predictions
Rigorously 'stress' all prediction in user testing
Confident coding of predictions as hits, false positives and 'misses'

Figure 1: Process of Falsification Testing

In summary, falsification testing is fixed task user testing. It is not open, users have restricted choice. Thus it cannot be used for accurate thoroughness scores, which are achieved via a complementary method of *asymptotic testing* [6]. Nor can the process distinguish between genuine misses and false negatives without evidence of analyst elimination (hence 'misses'). The goal of falsification testing is accurate validity scores, so task sets for testing are systematically derived from analyst predictions.

	Predicted Problem Description
1	Ambiguity of start screens regarding the selection of presentations and types of layouts
2	Toolbar command buttons small and hard to discriminate
3	Visual status of toolbar command button availability. No visual distinction between (un)available buttons
4	User has imited time to read error messages (5 secs)
5	Unfilled objects and 'white' filled objects are displayed the same, but selection and manipulation of these objects require different methods
6	Incorrect and misleading information on status bar when manipulating text
7	The task of ungrouping objects is not complete until the user performs a secondary unprompted action
8	Status Bar is badly positioned resulting in much of the system feedback being missed by the user
9	The user is forced to reselect the toolbar button
10	Cursor feed back, no change for circles or squares, remains as cross hairs
11	Graphic representation of toolbar buttons, hard to interpret
12	Right mouse button functions (shortcuts), hard to find

Table 2: Predicted problems from [2,12]

5.1 Example mapping

It is not always necessary to develop unique tasks for every predicted problem. Quite often individual tasks will address several predictions. Three tables from a previous study [2,12] show how tasks are mapped to specific usability problem predictions. Table 2 briefly describes 12 predicted usability problems. The study involved a 'drawing editor' (PowerPoint version 4.0), with which multiple analysts performed a heuristic evaluation. There were clearly many usability issues with selecting appropriate slides and layouts depending on the task at hand. Furthermore, issues with the drawing tools and manipulating objects were also of concern. In keeping to the context of use of the application, the general scenario involved a relatively inexperienced individual replicating a sketch in PowerPoint for an up-coming presentation.

After first of all recreating the drawing, the users were then required to *manipulate* various objects in order to address potential usability problems associated with such actions.

User test instructions in Table 3 specify tasks that address both specific and general potential problems (analyst predictions).

	User Task
a	Launch the application and create the drawing you have been supplied on a blank slide
b	The barrier in the drawing should be positioned at 60^o to the horizontal
c	The default setting for the colour of generated objects is blue. Do not change the colour of any of the objects until all objects have been created and positioned
d	Text sizes differ. Do not alter text size of any text until all of the text has been completed (exact font size is not critical, only maintain approximate proportionality).
e	Reposition the demonstration stands as shown in Drawing 2
f	Reposition the barrier to its new horizontal position
g	Remove stands 4, 8 and 11
h	There is a command button on the toolbar for generating squares. Create a box to surround the barrier and three of the display stands. Make sure that the information beneath it is visible. This can be achieved by selecting the 'no fill' option from the fill colour command on the toolbar
j	Reposition the square you have just created so that it surrounds the display stands

Table 3: User test instructions from [2,12]

Table 4 brings together the potential problem set and the instructions. The predicted problems are listed in the first column. The second column gives a brief description of the action that will expose the predicted problem. The third refers to the instruction(s) in Table 3 that, in the course of user testing, should cause users to encounter elements that were predicted as causes of potential usability problems.

Prediction	Exposing User Action	Task
1	Selecting desired slide format	a
2	Selecting various toolbar buttons	c,d,k
3	Selecting various toolbar buttons	c,d,g,k
4	Actions resulting in error messages	k
5	Need to manipulate filled and unfilled objects differently	e,g,h
6	Following incorrect information on status bar	d
7	Separate unprompted action required when ungrouping objects	e,g
8	Feedback on status bar often missed by user	b,f,k
9	Reselection of tool after one action	k
10	Limited cursor feedback	k
11	Perception of graphical representation of toolbar buttons	c,h,k
12	Right mouse button functions (shortcuts)	c,d,k

Table 4: Mapping from predictions to test instructions [2,12]

In Table 4, some problems are addressed by several actions, e.g., Problem 2, *Toolbar command buttons are small and difficult to discriminate*, is addressed by tasks:

(c) Changing object colours *after* they have been created

(d) Changing text size after all text has been inserted

(k) Wildcard.

A unique task is not always necessary to address potential problems. For example, no specific task was needed to address Problem 10 (limited cursor feedback). As test participants carried out various tasks designed to address other predictions, difficulties with cursor feedback would be apparent. **k** in Table 4 indicates such 'wildcard' steps.

5.2 Mapping and card sorts

Table 4 uses a 1:n mapping strategy from predictions to test tasks This was manageable due to the relatively small master problem set. With larger sets, a m:n mapping of several predictions to the same test tasks is possible. We used card sorts to create such compact mappings in a more recent study [3]. Predictions were sorted by the third author into groups that could be tested by the same (group of) test task(s). Larger problem sets can thus be handled without having a linear increase in test task set size.

6. CONCLUSIONS

Reliable assessment of UIMs depends on accurate coding of analyst predictions. Random user testing for UIM assessment carries the risk of miscoding. A combination of ESPRFs [4], asymptotic [6] and falsification testing (presented above) reduces the risks of miscoding, allowing for more accurate assessment of UIMs. These approaches increase confidence in the reliability (within their scope) of our studies [2,3,11,12]. Hopefully, sceptics on the existence of false positives will reconsider their position, and the true validity of UIMs will be acknowledged.

REFERENCES

[1] Cockton, G. & Lavery, D. "A Framework for Usability Problem Extraction", in INTERACT 99 Proceedings, eds. A. Sasse and C. Johnson, 347-355, 1999

[2] Cockton, G. & Woolrych, A., "Understanding Inspect-ion Methods: Lessons from an Assessment of Heuristic Evaluation," in People & Computers XV, eds. A. Blandford & J. Vanderdonckt, Springer, 171-192, 2001.

[3] Cockton, G., Woolrych, A., Hall, L., & Hindmarch, M., "Changing Analysts' Tunes: The Surprising Impact of a New Instrument for Usability Inspection Method Assessment" in People & Computers XVII, eds. O'Neill, Johnson & Palanque. Springer, 145-161, 2003

[4] Cockton, G., Woolrych, A. & Hindmarch, M. "Reconditioned Merchandise: Extended Structured Problem Report Formats in Usability Inspection" in Extended Abstracts of CHI 2004, ACM, 2004.

[5] Gray, W.D. & Salzman, M., "Damaged Merchandise? A Review of Experiments that Compare Usability Evaluation Methods", HCI, 13(3), 203-261, 1998

[6] Hartson, R.H., Andre, T S., Williges, R.C. "Criteria For Evaluating Usability Evaluation Methods". Int. J. of HCI, 15(1), 145-181, 2003

[7] Hertzum, M. & Jacobsen, N.E., "The Evaluator Effect: A Chilling Fact about Usability Evaluation Methods", Int. J. of HCI,13(4), 421-443, 2001

[8] Polson, P., Lewis, C., Rieman, J., & Wharton, C., "Cognitive walkthroughs: a method for theory- based evaluation of user interfaces," IJMMS, 36, 741-73, 1992

[9] Sears, A., "Heuristic Walkthroughs: Finding the Problems Without the Noise", International Journal of Human-Computer Interaction,9(3), 213-23, 1997

[10] Woolrych, A. and Cockton, G. "Assessing Heuristic Evaluation: Mind the Quality, not just Percentages" in HCI 2000: Vol 2, Eds. S. Turner and P. Turner, 35-36, 2000.

[11] Woolrych, A. and Cockton, G., "Testing a Conjecture based on the DR-AR Model of UIM Effectiveness," in *Proceedings of HCI 2002*, Vol 2, eds. Sharp, Chalk, LePeuple and Rosbottom, BCS, 30-33, 2002

[12] Woolrych, A. Assessing the Scope and Accuracy of the Usability Inspection Method Heuristic Evaluation, MPhil Thesis, University of Sunderland, UK, 20