Automatic Generation of High Coverage Usability Tests

Renée C. Bryce
Dept. of Computer Science & Engineering
Arizona State University
Tempe, AZ, USA
rcbryce@asu.edu

ABSTRACT

Software systems are often complex in the number of features that are available through the user interface and consequently, the number of interactions that can occur. Such systems are prone to errors when interactions do not work as anticipated. This research introduces a combinatorial method for setting up task-based usability tests. The method bridges contributions from mathematics, design of experiments, software test, and algorithms for application to usability testing.

Categories & Subject Descriptors: H.5.2**[User Interfaces]**: Evaluation/methodology

General Terms: Human Factors, Measurement, Theory, Verification

Keywords: Covering arrays, design of experiments, interaction testing, mixed-level covering arrays, usability testing

INTRODUCTION

A high coverage usability test is a form of interaction testing such that every combination of pair wise (or t-way) interactions is tested at least once. All pairs (or t-tuples) of user interactions are built into a collection of task variants called a test suite. Each task variant within the test suite is given to users for testing. The interaction coverage can enforce the execution of a well-rounded set of interactions that can occur in a system. As a supplement to traditional testing, this method can help to identify more usability problems. For instance, consider testing a screen of an on-line store as broken down in Table 1. During usability testing, one may choose to set up a suite of tests to exhaustively test all variations of tasks so that every scenario that may occur is reviewed by respective test users. In this example, this is 3^4(3*3*3*3), or 81 tests. Alternatively, a tester may choose a combinatorial approach such that each pair of interactions is covered at least once to get user feedback. This would reduce the 81 tests down to 9 tests as shown in Table 2.

Table 1: An example user interface has four features that each have three setting options

Log-in Status	Member Status	Discount	Shipping Method
New Member	Basic	None	Standard (5-7 day)
Existing Member – logged in	Silver	Employee	Express (3 - 5 day)
Existing Member – not logged in	Gold	Holiday	Overnight (1 day)

Table 2: There are 9 tests for pair-wise interaction testing

	Log-in Status	Member	Discount Status	Shipping Method
1	New Member	Basic	None	Standard (5-7 day)
2	New Member	Silver	Employee	Express (3-5 day)
3	New Member	Gold	Holiday	Overnight (1 day)
4	Existing Member – logged in	Basic	Holiday	Express (3-5 day)
5	Existing Member – logged in	Silver	None	Overnight (1 day)
6	Existing Member – logged in	Gold	Employee	Standard (5-7 day)
7	Existing Member – not logged in	Basic	Employee	Overnight (1 day)
8	Existing Member – not logged in	Silver	Holiday	Standard (5-7 day)
9	Existing Member – not logged in	Gold	None	Express (3-5 day)

Each test case results in different portions of the user interface being exposed. During any task-based usability test, with each interaction reviewed, doors are opened for the user to experience and comment on the respective features. For instance, if a user is a new member, they go through a registration page. If the user is a member that is not logged in, they are prompted to log on. Once logged on, they move on to the next step. In this example, it may be found that the "Silver" member status and "Employee" discount clashed causing a usability problem for the user. Assuming that this is an unlikely scenario that an average user most likely would not

CHI 2005, April 2–7, 2005, Portland, Oregon, USA.
ACM 1-59593-002-7/05/0004.

encounter, it could have been overlooked in a typical test that does not attempt to create the 81 tests. However, if all pairs of interactions are covered, the scenario will be accounted for.

This approach to setting up usability tests provides better coverage of features than simply telling the user to "place an order" and yet does not incur the high cost to execute numerous tasks for exhaustively testing all combinations of scenarios. Both standard and nonstandard usage patterns are examined in order to increase the assortment of usability problems that can be caught.

AN ALGORITHM FOR PRACTICAL CONCERNS

Constructing high coverage usability tests is not trivial. A combinatorial object called a mixed-level covering array must be constructed to adapt to the practical concerns of this application. Several methods for constructing these covering arrays have recently been published (the reader is referred to [3] for a brief summary of current methods). Many can quickly be applied for generating high coverage usability tests. The respective algorithms fall into one of three genres of algorithms: algebraic constructs, greedy methods, and heuristic search. In this application, algorithms are evaluated by the practical concerns and limitations that apply to the high coverage usability application.

In practical application, user interfaces are likely to have numerous features that take on various numbers of setting options. Testers want the most *accurate* (smallest size) test suite to be generated because every test adds to the cost of testing. A tester may also desire to seed part of the test suite with specific tests or partial tests that they want to be included in testing. Conversely, they may desire to specify constraints of combinations that do not make sense to combine in a test. *Consistency* of test suites is also desired. Test suites should not be randomly generated because it can cause confusion if different testers produce different test suites for the same problem. It should be possible for a tester to regenerate a test case that they had previously used. Further, test suites should be generated quickly so that testers do not have to wait a long time to begin their testing. Finally, user interfaces may change over time and hence a test suite should accommodate for change. This may include additions, removals, or modifications to features in a system. It may also include modification of the desired level of interaction coverage.

Currently, greedy algorithms may be the most appropriate construction techniques that are known to adapt efficiently to these practical concerns. The reader is referred to [1] for further support on this choice. We have pursued applying the Deterministic Density Algorithm (DDA)[2] to the practical concerns of this application. DDA may be considered a reasonable algorithm to select because it is deterministic, quick, provides competitive results, and guarantees that the size of the test suite is within a logarithmic size bound.

At the highest level of description, DDA constructs one row of a covering array at a time until all t-tuples are covered. Within the construction of each row, the order in which features are assigned values is based on density formulas. The formulas help to optimize the number of t-tuples that each row (or test case) will cover. For each feature, the value assigned is also selected by density formulas that calculate the likeliness of each value contributing to the most newly covered t-tuples in the row (or local test case). For an in-depth description of the algorithm, the reader is referred to [2].

ALGORITHM RESULTS

In practice, an exponential problem is reduced significantly in size when interaction testing is applied. For instance, Table 3 shows some sample problem parameters and sizes of test suites generated using DDA. The size of test suites included here is for pair-wise coverage. A tester may choose to have individual or separate users execute each task in the test suite.

Table 3: Sizes of test suites generated using DDA

Number of features	Number of settings for each feature	Size of test suite
4	3	9
40	2	13
100	2	14
11	1 with 5, 8 with 3, 2 with 2	21

FUTURE WORK

A contribution has been made with a new idea referred to as *high coverage usability testing*, definition of the method's context in usability testing, and the development of an algorithm to automatically and economically generate such test suites. Future work includes a vast amount empirical study of the application in practice and definition of its role in the User Centered Design process.

ACKNOWLEDGEMENTS

Thank you to my advisor, Dr. Charles J. Colbourn. This research is supported by the Consortium for Embedded and Internetworking Technologies (CEINT).

REFERENCES

1. R. C. Bryce, C. J. Colbourn, M. B. Cohen. A Framework of Greedy Methods for Constructing Interaction Tests. *Intl. Conf. on Software Engineering (ICSE'05), To appear.*

2. C. J. Colbourn, M. B. Cohen, and R. C. Turban. A deterministic density algorithm for pairwise interaction coverage. *Proc. of the IASTED Intl. Conference on Software Engineering*, pages 242–252, February 2004.

3. D. S. Hoskins, R. C. Turban, and C. J. Colbourn. Experimental Designs in Software Engineering: D-Optimal Designs and Covering Arrays. *Proc. SIGSOFT 2004/FSE-12: Workshop on Interdisciplinary Software Engineering Research*, pages 55 – 66, November 2004.