A Comparison of GUI and VUI Testing

Amber Wagner
Computer Science Department
University of Alabama
anwagner1@crimson.ua.edu

Abstract
Software testing is a critical step in the software development lifecycle, and it is equally important to evaluate the graphical user interface of an application. While there are many testing techniques used in testing graphical user interfaces, the number of techniques and studies for voice user interface testing is minimal. This paper presents an overview of selected techniques used in graphical user interface testing, which highlights that automated test case generation offers many benefits to the testing process. Voice user interface testing methods are also reviewed, and while there are some similarities to graphical user interface testing techniques, voice user interfaces do require different types of interaction. A comparison between the two testing groups is made along with the conclusion that the many methods of test case generation used for graphical user interfaces can be applied to voice user interfaces. This paper also proposes how a newly created graphical user interface and a newly created vocal user interface could benefit from selected testing methods presented within the paper.

1.0 Introduction
As Harris [9] points out, anyone who needs a justification for usability testing of an application requiring human interaction might want to change careers. Graphical User Interfaces (GUIs) are pervasive in software applications; it is the primary interface a user has with the software. 15 years ago, developers were dedicating over 50% of code to the GUI [18], and they require a great deal of testing. GUI testing, however, is not as pervasive. There are two categories of GUI testing: usability testing and functional testing [2]. Functional testing consists of four types: GUI system testing, regression testing, input validation testing, and GUI testing (does the software work) [2]. GUI testing in this paper refers to primarily the fourth type, but regression testing is touched upon. Often in GUI testing, not all event-sequences are adequately tested [18]. A sample of test cases may be selected, but one sample may produce one result and a second sample may produce a different result, thus, missing software faults [16]. To complicate testing further, new modalities have been introduced to user interfaces, such as voice.
Voice User Interfaces (VUIs) are similar to GUIs, but the only modality for interaction is a user’s voice. The system may respond back to the user vocally (audibly) or visually. The number of existing techniques for testing VUIs [3, 5, 6, 7, 8, 9, 10, 11, 12] is an order of magnitude less than what exists for GUIs. Many techniques for improving the testing of GUIs involve efficiently creating more complete test suites [1, 4, 13, 14, 15, 18, 19]. It is possible that some of these same techniques can be applied to VUIs in order to allow for more efficient and thorough evaluations.

This paper discusses existing GUI testing techniques (Section 2), existing VUI testing techniques (Section 3), compares and discusses possible overlap of GUI and VUI testing (Section 4), provides future work of evaluating both a GUI and a VUI (Section 5), and concludes with final thoughts (Section 6).

2.0 GUI Testing Techniques

The critical component in testing software is to develop a comprehensive test suite. GUIs have an exponential amount of possible interactions requiring a very large number of test cases in order to conduct comprehensive testing on how well the GUI works; however, a common type of tool used in GUI testing, capture/replay tools, results in creating only a small number of test cases causing insufficient testing [18]. In addition to the challenge of creating test cases, the test cases must be maintained and executed; this is where automation plays an important role as maintaining and executing the test suite is expensive [13, 14, 15, 18]. Although there are a variety of techniques for maintaining and executing test suites, selected techniques are discussed below in two categories: regression testing and fault detection. A recurring theme throughout the research is that test cases should be generated automatically (minimizing human interaction) [1, 14, 18].
2.1 Regression Testing

A key component of using regression testing is to have an accurate test suite, but as previously mentioned, the test suite can be difficult to maintain [13, 19]. Maintenance can be complicated because GUIs are commonly developed using rapid prototyping [13], which involves many iterations of a GUI resulting in unusable tests within the test suite. Updating the test suite can become expensive; thus, using an automated approach for repairing the test suite, like that presented in [13], could be beneficial. Memon [13] presents an automated approach for repairing the unusable tests allowing a tester to then rerun these tests on the newly modified GUI. Similarly, Yin et al. [19] present an Actionable Knowledge Model utilizing an agent based framework to maintain and repair test suites. Another issue with the regression testing of GUIs revolves around software being updated nightly. Often, the GUI testing of nightly updates is forgotten [14]. DART (Daily Automated Regression Tester), an automated regression testing framework, addresses this issue [14]. DART allows GUIs to be smoke tested (initial testing after modification to ensure application works) automatically on a frequent basis and at a low cost [14].

2.2 Fault Detection

Similarly to regression testing, the key component in fault detection in GUIs is an adequate test suite. Fine tuning the test suite through prioritization [4] or refining existing models [18] allows for improved GUI testing. Bryce and Memon [4] evaluated test suite prioritization using different methods (e.g., unique event coverage, Event Interaction Coverage (EIC), longest to shortest, shortest to longest, and random test ordering) finding that using EIC to prioritize the test suite “increases the rate at which faults are detected during GUI testing.” In a later work, Xie and Memon [18] refine the event-flow graph (EFG) creating an event-interaction graph (EIG), which is reduced using minimal effective event context (MEEC) thereby creating fewer test cases. Using the resulting test suite, Xie and Memon [18] evaluated the EIG test suite using four
open source software applications, which found bugs not previously reported yet still relevant; thus, the EIG created test suite provided a more comprehensive test suite than those previously utilized.

3.0 VUI Testing Techniques

There are a variety of testing techniques utilized in industry including Dialog Traversal Testing (DTT), system quality assurance testing, load testing, recognition testing, evaluative usability testing, Wizard of Oz (WOZ) testing, usability inspections, VUI Review Testing (VRT), surveys, call recordings, and call logs [3, 5, 6, 7, 8, 9, 10, 11, 12]. These evaluation techniques are used at different phases during the development life cycle; however, system designers apply the majority during the design or evaluation phase of the development life cycle, only surveys, call recordings, and call logs are exercised after deployment. Four techniques (e.g. Wizard of Oz testing, VUI review testing, usability inspection, and prototype testing) are discussed in more detail in the following sections.

3.1 Wizard of Oz Testing

“The Wizard of Oz is not only a classic movie, but also a testing methodology through which you can tell your application is on the right road to Kansas” [3]. Wizard of Oz (WOZ) testing is a very common testing technique used primarily during the design phase of the development life cycle [6, 8, 9, 12], but it can also be useful to a developer during the test and analysis phases [11]. It consists of a “wizard” who reads system prompts based on the call flow design [8]. Similar to prototype testing [9, 12], WOZ testing allows designers to evaluate the program flow (i.e., test call flow, grammar, prompts, etc.) without investing a significant amount of resources into the project [8].

3.2 Prototype Testing

While similar to WOZ testing, prototyping takes testing to the next level as it involves using an actual system prototype [12], which implies that coding has been performed. It is
important to test using a functioning prototype and has been referred to as the “gold standard” in evaluating an application’s usability and results in the amount of data collected being “an order of magnitude more realistic than the data from a WOZ test” [12]. Kortum [12] expresses the opinion that WOZ testing is more resource intensive than at first glance and that the “benefit is exaggerated” relative to the amount of work it requires as compared to the cost-benefit of prototype testing.

3.3 VUI Review Testing

VUI Review Testing (VRT) is a type of “holistic” and “experiential” review where a VUI designer reverses roles and acts as the caller of the system exercising each use-case scenario previously determined [10]. It is important that the tester be a VUI designer or usability specialist and not a developer because the goal of VRT is to verify the quality of the user experience [10]. VRT should be conducted after DTT has already been performed but before User Acceptance Testing (UAT) in order to catch possible usability problems, prompt quality problems, and to determine if and where pauses might need to be placed or extended in addition to other, more general, problems [10]. Because VRT catches these types of issues, it increases the informativity of usability testing.

3.4 Usability Inspection

Usability inspections are more empirical in nature and, thus, more objective [6, 7, 9]; however, the number of established metrics for evaluating VUIs is minimal [12]. Dybkjær and Bernsen [6] created a template consisting of 15 usability issues, which the usability of the VUI to be evaluated on specific criteria, but many of these criteria are evaluated subjectively rather than objectively. Farinazzo et al. [7] also present a checklist type methodology for a heuristic evaluation. In both [6] and [7], a usability expert should the one performing the evaluations. Evaluation criteria resulting in more empirical/objective data might consist of the number of interaction problems, proper entry recognition rates, mean recognition scores, task success
ratio, timeouts, number of recognition by type (rejection, deletion, insertion, substitution), early terminations, etc. [6, 7, 9].

4.0 GUI Testing vs. VUI Testing

While some VUI testing techniques are similar to those in GUI testing, VUI testing presents new challenges due to the temporary nature of the voice input modality. Many of the existing techniques for VUI testing focus on Interactive Voice Response (IVR) systems, which are the telephone systems companies use to manage customer calls. These vary greatly from a GUI where the input modality is voice rather than the mouse/keyboard. While the input modality is temporary, the feedback is permanent. This type of VUI is a mutant of a GUI and the VUI referenced in the aforementioned VUI testing techniques. The primary difference between GUI and VUI testing is that GUI testing appears to be more empirical (objective) [1, 4, 13, 14, 15, 18, 19] where VUI testing appears to be more user-focused (subjective) [3, 5, 6, 7, 8, 9, 10, 11, 12]. VUI testing would benefit from more objective testing [6, 7, 9]. A method to add more objectivity to VUI testing would be to take advantage of existing GUI techniques. Based on the research previously discussed, this would mean developing more comprehensive test suites, automatically, which would evaluate the success of the application based on set requirements rather than subjectively.

No matter the technique borrowed from GUI testing, WOZ testing and prototype testing would still be necessary to capture the vocal portion of the application; although, WOZ testing in this mutant VUI is a minimized version. Prototype testing would be more appropriate than WOZ testing other than to evaluate phrases during the design phase. VRT and usability inspections as described in Sections 3.3 and 3.4, respectively, would be helpful to ensure a fully comprehensive evaluation were performed, particularly from the user’s perspective. Any other testing performed would add to existing testing in order to examine the behavior of the mutant VUI.
One possible strategy for creating a test suite might be to use the commands in the grammar as if they were buttons and textboxes. The grammar would have to be written in such a way that commands (e.g., file, edit, new, close, etc.) would be separated from terminals (e.g., number, 0, letter, a, etc.). Moreover, the grammar should be written so that a parser could depict the type of each verbal command (i.e., distinguish a menu from a menu item). For example, the following could be a subset of a grammar where the word on the left-hand side of each row (following an action command) represents the menu, and any words thereafter represent a menu item.

```java
public <click_action> = <click> (file | new | open | save | quit |
    edit | copy | cut | paste);
```

An EIG [18] could then be created and a comprehensive test suite developed. Figure 1 represents the EIG for the Open menu item. Once the test suite is created, it could then be prioritized using event interaction (since an EIG would be used to create the test suite, there may not be more prioritization necessary) [4] possibly reducing the number of test cases. The ability to repair test suites for regression testing would be important, and the technique in either [13 or 14] could be used. If nightly updates were necessary for the VUI, DART [14] would be the most appropriate tool. Figure 2 depicts a possible development lifecycle for a VUI with testing components added.
5.0 Future Work

Myna is a framework that allows developers to create VUIs for GUI-based Initial Programming Environments (IPEs) [17]. The resulting VUIs have thus far not been formally
evaluated. Additionally, MynaMapper, a GUI used within the Myna framework should also be formally evaluated.

5.1 Applying GUI Testing to MynaMapper

MynaMapper is a small GUI created as part of the Myna framework. It is used to map the coordinates of components within an IPE to property files that Myna will later reference. MynaMapper requires a great deal of input from the user (i.e., the user must click and drag various components on the screen). It would be interesting to utilize an EIC [18] to create a test suite. Other techniques, such as [13, 14, 19] are not as applicable since MynaMapper does not undergo many modifications.

5.2 Applying VUI Testing to Myna

Although the Myna framework has moved well beyond the design phase, the resulting VUIs have not. There are numerous IPEs for which Myna could be used to develop a VUI, and some of the techniques discussed in Section 3 could be used during the design and testing phases of the development life cycle. Because WOZ testing is primarily for IVR systems, the effort it takes to set up a WOZ test does not seem worth the benefit; however, something similar where the grammar could be evaluated during the design phase would be important. VRT and usability inspections would be very applicable and should be performed. VRT the testing of use-case scenarios, which would be easy to apply to a VUI created using Myna. Empirical data from usability inspections (e.g., recognition rates, success rations, timeouts, etc.) would provide an objective evaluation of the resulting VUI. Moreover, creating test cases using an EIG [18] would provide an additional objective evaluation of the VUI. A system for running these types of evaluations should be added to the Myna Framework for future developers to utilize.

6.0 Conclusion

This paper described selected GUI and VUI testing techniques and discussed how GUI testing techniques might be combined with VUI testing to create a more formal testing
methodology for VUIs since most VUI testing is subjective. GUI testing focuses on the automatic creation and maintenance of comprehensive test suites where VUI testing focuses more on user-testing. Creating test cases for VUIs using those same methodologies for creating test cases for GUIs would allow for a more empirical evaluation of a VUI; although traditional VUI testing must still be performed in order to ensure the grammar, and the modality in general, are appropriate. Moving forward, it would be interesting to evaluate MynaMapper using test cases developed via the GUI techniques described (EIG [18] is the most applicable), and the VUIs created using the Myna framework should be evaluated by the proposed strategy in Section 4.

Acknowledgements

This project is supported by NSF grant IIS-1117940, with additional support from a Google CS4HS Award. I would like to thank Dr. Gray for his advice in preparing this paper and Raymond Wagner for reviewing this paper.

References


