Revisiting the Notion of GUI Testing

Abdulaziz Alkhalid, Yvan Labiche and Sashank Nekkanti
Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada

Keywords: System Testing, GUI Testing.

Abstract: The practitioner interested in reducing software verification effort may found herself lost in the many alternative definitions of Graphical User Interface (GUI) testing that exist and their relation to the notion of system testing. One result of these many definitions is that one may end up testing twice the same parts of the Software Under Test (SUT), specifically the application logic code. We revisit the notion of GUI testing and introduce a taxonomy of activities pertaining to testing GUI-based software. We use the taxonomy to map a representative sample of research works and tools, showing several aspects of testing GUI-software may be overlooked.

1 INTRODUCTION

Advances in technology used as platforms for Graphical User Interface (GUI) software lead to more complex, platform-independent GUI-based software. Current GUI software are capable of serving different types of users with different levels of abilities (e.g. ordinary user, user with disability, Web user, or Mobile user). These advances in technology produce challenges for software testers who are responsible for software verification of those GUI-based software. One of them is that software testers find themselves in front of several testing types to choose and use, such as GUI testing and system testing.

A well accepted definition of software system testing is that it is a phase of software testing conducted on the complete software to evaluate its compliance with its requirements, be they functional or non-functional (Desikan and Ramesh 2006). However, there is a confusion about alternative definitions of GUI testing one can find in the literature. For example, Ammann and Offutt classified GUI testing into usability testing and functional testing and further classified the latter into GUI system testing, regression testing, input validation testing and GUI testing (Ammann and Offutt 2008). They argue that GUI system testing is system testing of the entire software through its GUI while GUI testing is verifying that the GUI works correctly without verifying the underlying application code. Memon et al. defined GUI testing as system testing for software that has a graphical user interface (Banerjee, Nguyen et al. 2013). We conclude that Memon's notion of GUI testing encompasses both notions of GUI testing and GUI system testing of Ammann and Offutt. One could argue that these are only two authors and that they may not be representative. We conclude that there is no agreement about the notion of "GUI testing", about what it is means and what it entails.

As further shown by our study of literature on the topic, we conclude that the reader interested in testing a GUI-based software may found herself lost in the many alternative definitions of GUI testing that exist and their relation to the notion of system testing. For instance, using Memon's definition of GUI testing, one can use a tool like GUITAR (Memon 2015) to trigger both the GUI and the underlying functionalities whereas when using Ammann and Offutt's definitions one can use JUnit to directly test the application code, bypassing the GUI, and verify the GUI separately. One risk of using incompatible definitions for GUI testing and system testing is to duplicate testing effort: One conducts system testing of the application logic by bypassing the GUI and conducts GUI testing of the software with GUITAR (Nguyen, Robbins et al. 2014), thereby testing the application logic twice. This paper therefore attempts to answer the following research question: What are available definitions of system and GUI testing and how they relate to each other?

Alkhalid, A., Labiche, Y. and Nekkanti, S.
Revisiting the Notion of GUI Testing.
DOI: 10.5220/0006901606400647
In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*, pages 640-647
ISBN: 978-989-758-320-9

Figure 1 illustrates the focus of this paper. It illustrates several software testing definitions by showing the software divided in its GUI layer and its application logic layer.

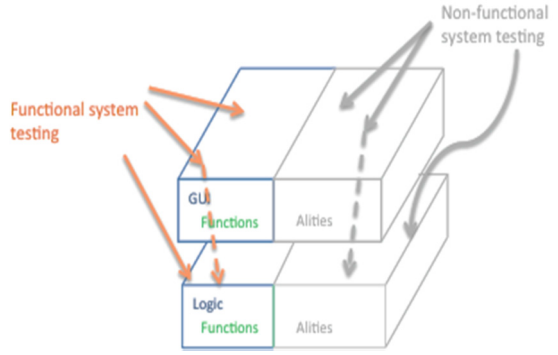

Figure 1: Functional and non-functional system testing

It illustrates that system testing can focus on the functional aspects of the System Under Test (SUT), referred to as functional system testing, or the non-functional aspects of the SUT also sometimes referred to as the "alities", referred to as non-functional system testing. Both can trigger only the GUI (an arrow stops at the GUI layer, meaning that the application logic is being stubbed/mocked), the GUI and the underlying application logic layer (arrow to the GUI layer, going through the GUI as dashed line and triggering the application logic layer) or only the application logic layer. It also shows that our scope, non-greyed-out part, is limited to functional system testing and does not deal with the alities of the SUT. Although our contribution includes some discussion of the notion of testing the alities of a GUI-based software, we decided to focus on functional aspects rather than non-functional ones. Additionally, the majority of related work and available tools also focus on functional aspects rather than non-functional ones. As justified later in the paper and illustrated in the figure, when functional system testing is applied through the GUI, we call it GUI system testing in order to distinguish it from functional system testing applied to the application logic directly. We focus on GUI applications since such applications typically require robust UIs (Forrester and Miller 2000; Ganov, Killmar et al. 2008). Another motivation is the difficulty, to the point of impracticality, of GUI system testing for any SUT with non-trivial UI: for instance, using GUITAR (Memon 2015) on Microsoft WordPad in Windows 7 (Nguyen, Robbins et al. 2014), which contains over 50 GUI events, is extremely expensive (in terms of number of tests).

The rest of this paper is structured as follows. Section 2 describes our search method. Section 3 describes definitions of system testing and GUI testing. Section 4 presents our definitions and taxonomy. Section 5 uses the taxonomy to map a representative sample of existing research activities and tools on testing of GUI-based software. Section 6 presents conclusions.

2 OUR SEARCH METHOD

We present some definitions of system testing, GUI testing, and other testing activities. As discussed below, these definitions warrant the study of differences (if any) between system testing and GUI testing. The intent of this paper is not to report on a systematic mapping study on GUI testing definitions and other testing definitions. We simply report on representative definitions of main software testing terms to answer research question: What are available definitions of system and GUI testing and how they relate to each other?

We used a systematic method, though not a systematic literature review or systematic mapping study, to identify relevant definitions. The method started by identifying textbooks in our possession or at the University Library in the area of software engineering and software testing. In the case of library books, this meant using the Library search engine to identify books using the following keywords: testing, software GUI testing, software verification, GUI testing. Then, we identified chapters of those books which discuss software testing and in particular GUI testing by browsing through the tables of contents and skimming through pages, looking for keywords like "GUI testing" or "system testing". We identified a total of 52 textbooks (Alkhalid and Labiche 2016). We believe that, for our search for definitions, looking into textbooks is an adequate procedure, rather than for instance searching in academic paper databases. We nevertheless surveyed by searching online resources too, i.e., Google Scholar, IEEE Xplore, Science Direct, ACM, Engineering Village and Scopus using the following search strings: Graphical User Interface Testing, GUI testing, GUI testing "AND" system testing, definition of GUI testing, Oracle for GUI testing, GUI testing tools, automated GUI testing, survey of GUI testing, GUI testing taxonomy. This step was necessary to find recent surveys or taxonomies in the area of GUI testing. This allowed us to identify a recent (2013) systematic mapping study on GUI testing (Banerjee, Nguyen et al. 2013)

which we later use when mapping existing work with our taxonomy. We used the dblp Computer Science Bibliography (Ley 1993) to look for publications on GUI testing when needed for a specific author.

3 SYSTEM AND GUI TESTING

System testing is defined as a "testing phase conducted on the complete integrated system to evaluate the system compliance with its specified requirements on functional and non-functional aspects" (Desikan and Ramesh 2006). GUI testing can be defined as system testing for software that has a GUI (Banerjee, Nguyen et al. 2013; Nguyen, Robbins et al. 2014), that is system testing of the entire software performed through its GUI. Assuming the standard, IEEE definition of system testing we already discussed, we argue that GUI testing as defined by Memon creates tests that do not (necessarily or specifically) address "alities". According to Ammann and Offutt, determining whether the GUI and the logic of a GUI-based software behave as expected 1 includes usability testing and functional testing (Ammann and Offutt 2008). The former refers to the assessment of how usable the interface is according to principles of user interface design. The latter refers to whether the user interface works as intended. They further classified functional testing in this context into four categories: GUI system testing, regression testing, input validation testing and GUI testing. GUI system testing refers to "the process of conducting system testing through the GUI". Regression testing is about "testing of GUI after changes are made" (Ammann and Offutt 2008). Input validation testing aims to verify whether the GUI "recognize[s] the user input and respond[s] correctly to invalid input" (Ammann and Offutt 2008).

We first notice that Ammann and Offutt's definitions do not account for alternative non-functional requirements of the UI to usability and robustness (input validation), which also need to be verified. Contrasting Amman and Offutt's definition to Memon's definition, we see that Memon's notion of GUI testing is identical to the notion of GUI system testing by Ammann and Offutt, except with regards to some non-functional requirements. The top part of Figure 2 illustrates the main definitions we have encountered in our survey and that we just discussed (the "orange" arrows are discussed next).

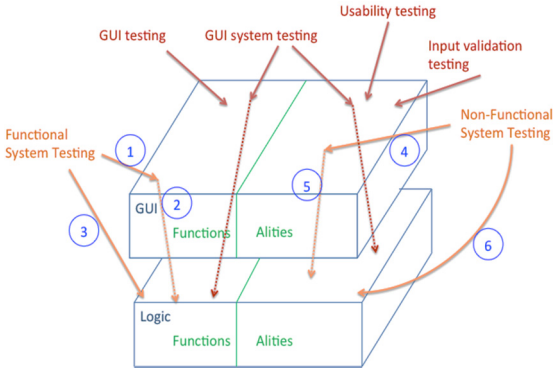

Figure 2: Relationship between different testing types.

In red the figure illustrates Ammann & Offutt's definitions. GUI testing is about the functional aspects of the GUI, focusing only on the UI layer, so the arrow goes to the functional part of the GUI and stops there.

From their definitions, we do not have evidence that GUI testing also focuses on non-functional characteristics, especially since usability testing is a separate activity in their discussion. Usability testing is about an "ality" so the arrow goes to the "alities" part of the GUI and stops there. GUI system testing is system testing through the UI so arrows go to the UI (both functional and non-functional) and go through to the application logic.

4 A TAXONOMY OF TERMS PERTAINING TO TESTING OF GUI-BASED SOFTWARE

In this section, we present an initial, therefore likely incomplete, taxonomy of terms that pertain to testing of a GUI-based software. In line with the majority of the references on the topic, including the IEEE definition, we abide by the definition that states that system testing is about evaluating compliance of an entire software system with its specified functional and non-functional requirements. It follows that, although prominent definitions of system testing (Abbott 1986; Lewis 2004; Desikan and Ramesh 2006; Homes 2012) do not explicitly mention the GUI, in case the software system has a GUI, system

[1] Ammann and Offutt discuss that usability testing and functional testing are the two activities of GUI testing. They then split functional testing into four categories, including GUI testing, which results in a circular definition of the notion of GUI testing. We believe this circular definition was not intentional. To avoid this circular definition, we write that usability testing and functional testing are the two activities involved in determining whether the GUI and the logic of a GUI-based software behave as expected.

testing encompasses the evaluation of the GUI against (GUI-specific) functional and non-functional requirements because system testing works on the entire product. This confirms that system testing includes GUI testing, which is very much like, though slightly different to, Ammann & Offutt definition, as discussed earlier. A direct consequence of this statement is that system testing and GUI testing are two different things and that GUI testing cannot be system testing applied on the GUI.

Figure 2 is slightly different from Figure 1 and illustrates the general definitions of system testing we abide to (orange arrows): directly exercising the UI or the application logic layers (direct, plain arrows), possibly exercising the latter through the former (dotted arrows). Those tests focus on either functional or non-functional characteristics, which we refer to as *functional system testing* and *non-functional system testing*, respectively.

We define *functional system testing* as checking conformance of the entire GUI-based software against its functional requirements, either by directly interacting with the application logic (arrow 3 in Figure 2), by isolating (stubbing/mocking the application logic) and focusing only on the UI (arrow 1), by focusing on the UI in combination with the application logic (arrows 1 and 2), or a combination of those.

We also define *non-functional system testing* as checking conformance of the entire GUI-based software against its non-functional requirements, either by directly interacting with the application logic (arrow 6), by isolating and focusing only on the UI (arrow 4), by focusing on the UI in combination with the application logic (arrows 4 and 5), or a combination of those.

GUI system testing can be either functional or non-functional, thus we use the terms *GUI functional system testing* for arrows 1+2 and *GUI non-functional system testing* for arrows 4+5. GUI functional system testing therefore encompasses system level tests exercising the entire software, that is through its UI, and checking conformance with both GUI-specific functional requirements and application logic-specific functional requirements. GUI non-functional system testing encompasses system level tests exercising the entire software, that is through its UI, and checking conformance with both GUI-specific and application logic-specific non-functional requirements.

We also refer to system testing of the application logic code, whereby the UI is bypassed, to as *functional system logic testing* (arrow 3) and *non-functional system logic testing* (arrow 6). Functional system logic testing therefore encompasses system level tests that specifically check conformance of the application logic code with application logic-specific functional requirements.

Non-functional system logic testing encompasses system level tests that specifically check conformance of the application logic code with application logic-specific non-functional requirements.

We also call *GUI functional testing* the testing of the functional aspects of the UI that does not require the application logic (arrow 1 only), and *GUI non-functional testing* the testing of the non-functional aspects of the UI that does not require the application logic (arrow 4 only). In both cases the application logic is stubbed/mocked. Therefore, GUI functional testing encompasses system level tests that specifically check the conformance of the UI part of the software (and only the UI) against UI-specific functional requirements. And GUI non-functional testing encompasses system level tests that specifically check the conformance of the UI part of the software (and only the UI) against UI-specific non-functional requirements. These are specific, focused version of the notions of functional system testing and non-functional system testing discussed earlier.

One general issue with software testing is how to provide the right values to the software. Software controllability describes how easy it is to provide a program with the needed inputs, in terms of values, operations, and behaviours (Ammann and Offutt 2008). For example, it is easy to control a piece of software for which all inputs are values entered from a keyboard (Freedman 1991; Gao 2000; Ammann and Offutt 2008). On the other hand, when the software gets its input values from sensors, it may be difficult to control. Typically, a tester has less control with component/system testing than with unit testing. Therefore, controllability can also mean the ease to reach some predefined level of coverage, i.e., to exercise specific behaviour or pieces of code: it is more difficult to reach coverage of units with system testing than with unit testing. In general, with a higher level of testing (e.g., system testing) it is harder to trigger specific elements of the code/functionality provided by lower levels of the code than with a lower level of testing (e.g., unit testing). When doing integration testing, it is harder to trigger specific statements of the code than with testing those units of the code directly. Similarly, when doing GUI functional system testing (arrows 1 plus 2 of Figure 2), it is harder to trigger code elements or behaviour of the application logic than

Table 1: Classification Results.

Ref	Primary	Secondary	Ref	Primary	Secondary
(Kepple 1992)	1+2	1 +2	(Derezinska and Malek 2007)	1+2	1+2
(Li, Huynh et al. 2007)	1+2, 3	1+2, 3	(Yuan and Memon 2010)	1+2	1+2, 4+5
(Mateo Navarro, Sevilla Ruiz et al. 2009)	1+2	1+2	(Yuan, Cohen et al. 2011)	1+2	1+2
(Tsujino 2000)	1+2	1+ 2	(Yang, Chen et al. 2014)	1+2	1+2
(Takahashi 2001)	1	1	(Memon, Pollack et al. 2001)	1+2	1+2
(Memon 2007)	1+2	1+2	(Mao, Boqin et al. 2006)	1+2	1+2
(Ye, Feng et al. 2007)	1+2	1+2	(Chen, Tsai et al. 2005)	1+2	1+2
(Memon 2008)	1+2	1+2, 4+5	(Yuan and Memon 2010)	1+2	1+2
(Memon, Nagarajan et al. 2005)	1+2	1+2, 4+5	(Chen and Subramaniam 2002)	1+2	1+2
(McMaster and Memon 2008)	1+2	1+2	(Memon and Xie 2005)	1+2	1+2
(Karam, Dascalu et al. 2006)	1+2	1+2	(Pham, Holzmann et al. 2014)	1+2	1+2
(Xie and Memon 2007)	1+2	1+2	(Xie and Memon 2008)	1+2	1+2
(Memon 2006)	1+2	1+2	(Alsmadi 2013)	1+2	1+2
(Alsmadi, Samarah et al. 2011)	1+2	1+2			

when doing functional system logic testing while bypassing the UI (arrow 3 in Figure 2), and even more so than when doing unit testing. This is another reason that helps justify the distinctions we make between the different testing activities mentioned earlier and illustrated in Figure 2.

5 MAPPING EXISTING WORK WITH OUR TAXONOMY

In this section, we use the new terms we introduced to map existing research. To do that, we look for primary studies in literature. We found a recent (published in 2013) systematic mapping study of GUI testing techniques (Banerjee, Nguyen et al. 2013). As an initial study we sampled the list of references Banerjee et al. classified and selected the 29 journal papers they identified. Two of them, references (Rubel and Quitslund 2007) and (Janicki, Katara et al. 2012), are surveys and cannot be mapped with our taxonomy, resulting in 27 studies to map. We selected journal papers since they admittedly represent the most developed research activities in the field. To classify a paper, we follow a set of steps: (1) We study the testing technique presented in the paper; (2) We identify the testing objective of that technique and classify it as one or more of the testing types we introduced earlier (arrows 1, 2, 3, 4, 5, 6 in Figure 2); (3) We analyze, based on our own judgment, whether the technique potentially (though this is not the primary purpose) covers other elements of our taxonomy. Table 1 shows the results of our classification (short justifications available in our technical report (Alkhalid, Labiche et al. 2018)). For each referenced journal paper (1st/4th column), the table indicates the primary purpose of the work in terms of arrows in Figure 2 (2nd/5th columns) and potential purposes (3rd/6th columns). We do not use the terminology in the table for space reasons. Results show that the vast majority of works (25, 93%) do GUI functional system testing (arrows 1+2). Only one study does GUI functional system testing and functional system logic testing (arrows 1+2, and 3), and only one study does GUI functional testing (arrow 1).

We did not find any work that specifically focuses on non-functional aspects (Primary objective). Only Memon and colleagues, with GUITAR, incidentally achieve more than their Primary objective, which is GUI functional system testing (Memon, Nagarajan et al. 2005; Memon 2008; Yuan and Memon 2010). This is due to the fact that their tool, GUITAR, can be used to provide erroneous inputs to the GUI under test: there is some GUI non-functional system testing.

In addition to published academic papers, we briefly characterize a number of GUI testing tools according to our taxonomy of testing activities (Alkhalid, Labiche et al. 2018). Our main focus is tools that support Java: in an initial use of our taxonomy, we wanted to scope the search for tools and Java is a popular programming language for tool development. Characterizing a specific tool according to our taxonomy involve either one or more of the following activities: (1) reading available, online documentation about the tool; (2) downloading (possibly a trial, time-limited) version of the tool and reading documentation that comes with it; (3) trying the downloaded tool on a case study.

Whenever possible we performed the classification based on evidence, which is either documentation that can be referenced, or experimental results obtained by using a tool on a case study. We selected 18 tools out of a set of 39 documented GUI testing tools (Desyatnikov 2016), making sure we have variety: freeware, open source, commercial. The procedure we used to select the tools was the following: We scanned the documentation or downloaded the tools; we excluded the ones with unavailable download (e.g. the download link was broken); we excluded commercial ones for which we cannot get a student or a limited license as well as the ones which do not support Java. The results Table 2 shows our results of classification of tools. All the tools do GUI functional system testing except Fitnesse and LoadUI as both of them are dedicated to non-functional aspect of software.

Table 2: Classification of Gui Testing Tools.

Tool	1	4	1+2	4+5	3	6
Abbot (Java apps only)	Yes	No	Yes	Yes	Yes	No
Fitnesse	No	No	No	No	Yes	No
Sikuli	Yes	Yes	Yes	Yes	No	No
SWTBot (SWT apps only)	Yes	No	Yes	Yes	Yes	No
Jubula	Yes	No	Yes	No	Yes	No
GTT (Java Swing apps only)	Yes	No	Yes	Yes	Yes	No
PowerShell Extensions (Windows apps only)	Yes	No	Yes	Yes	No	No
AutoIt (Windows platform only)	Yes	No	Yes	Yes	No	No
Maveryx (Java&Android only)	Yes	No	Yes	Yes	Yes	No
Selenium	Yes	No	Yes	Yes	No	No
Sahi	Yes	No	Yes	Yes	No	No
Cucumber#	Yes	No	Yes	Yes	Yes	No
Cubic test	Yes	No	Yes	Yes	No	No
EggPlant	Yes	Yes	Yes	Yes	No	No
Ranorex#	Yes	No	Yes	Yes	Yes	No
LoadUI	No	No	No	No	Yes	Yes
Squish#	Yes	No	Yes	Yes	Yes	No
SilkTest#	Yes	No	Yes	Yes	Yes	No

6 CONCLUSION

Recognizing there exist ambiguities around definitions of GUI testing, we presented a taxonomy of terms that distinguishes testing of a GUI-based software along two dimensions: whether functional or non-functional aspects are specifically targeted; whether tests exercise the UI only, the UI and the application logic together, or only the application logic.

We evaluated a select number of most developed related works and tools against this taxonomy and reported that the vast majority of works look alike in light of the taxonomy: they conduct what we coined GUI functional system testing, which is system testing through the UI of functional aspects of the entire application, that is functional aspects of the UI as well as functional aspects of the application logic. We first note that our definitions help distinguish functional aspects of the UI from functional aspects of the application logic. It appears from our mapping that existing works and tools primarily focus on functional aspects of the application logic, through the UI, and not necessarily on functional aspects of the UI.

We also note that controllability issues are not discussed in these works and tools. Specifically, controllability issues may arise and prevent achieving all objectives in terms of functional testing of the application logic code through the UI, which calls for additional system level testing of the application logic which we coined functional system logic testing.

Our results also show that very few works and tools consider non-functional aspects of a GUI-based software, i.e. both non-functional characteristics of the UI as well as non-functional characteristics of the application logic. Very few of them distinguish the (testing) verification of the UI from the (testing) verification of the application logic, despite the fact that, according to our discussion and according to standard software design principles, the two might be different. For instance, it is conceivable to observe a GUI-based software that passes verification conditions (functional and non-functional) established for the application logic but fails to pass verification conditions (functional or non-functional) established for the UI.

Acknowledging our taxonomy may be debated, we believe it is nevertheless a good starting point to continue the discussion as to what we should call “GUI testing”. We argue the taxonomy is useful to have an overview of the field and pave the way to future work: e.g., do GUI functional system testing (arrows 1+2) and functional system logic testing (arrow 3) exercise functionalities differently? How is this related to controllability, if ever? What GUI functional testing, the testing of the functional aspects of the UI without the application logic (arrow 1) look like? What about specifically focusing on non-functional aspects?

REFERENCES

Abbott, J. (1986). Software testing techniques, NCC.

Alkhalid, A. and Y. Labiche (2016). Comparing GUI System Testing with Functional System Testing-An Experiment. *Technical Report TR-SCE-16-01. Ottawa, Carleton University*.

Alkhalid, A., Y. Labiche, et al. (2018). Revisiting the notion of GUI testing. *Technical Report TR-SCE-18-02. Ottawa, Carleton University*.

Alsmadi, I., S. Samarah, et al. (2011). Evaluate and Improve GUI Testing Coverage Automatically, *IJSE*.

Alsmadi, I. M. (2013). "Using Mutation to Enhance GUI Testing Coverage." *IEEE software 30(1): 67-73*.

Ammann, P. and J. Offutt (2008). Introduction to Software Testing. *New York, Cambridge University Press*.

Banerjee, I., B. Nguyen, et al. (2013). "Graphical user interface (gui) testing: Systematic mapping and repository." *Information and Software Technology 55(10): 1679-1694*.

Chen, J. and S. Subramaniam (2002). "Specification-based testing for GUI-based applications." *Software Quality Journal 10(3): 205-224*.

Chen, W.-K., T.-H. Tsai, et al. (2005). Integration of specification-based and CR-based approaches for GUI testing. *Advanced Information Networking and Applications, 2005. AINA 2005. 19th International Conference on, IEEE*.

Derezinska, A. and T. Malek (2007). "Experiences in Testing Automation of a Family of Functional-and GUI-similar Programs." *IJCSA 4(1): 13-26*.

Desikan, S. and G. Ramesh (2006). Software testing: principles and practice. *India, Pearson Education*

Desyatnikov, R. (2016). "Top 39 GUI Testing Tools List." Retrieved 2016, from http://www.softwaretestinghelp.com/best-gui-testing-tools/.

Forrester, J. E. and B. P. Miller (2000). An empirical study of the robustness of Windows NT applications using random testing. *Proceedings of the USENIX Windows System Symposium*.

Freedman, R. S. (1991). "Testability of software components." *IEEE Transactions on Software Engineering 17(6): 553-564*.

Ganov, S. R., C. Killmar, et al. (2008). Test generation for graphical user interfaces based on symbolic execution. *Proceedings of the 3rd international workshop on Automation of software test, ACM*.

Gao, J. (2000). Component testability and component testing challenges. *Proceedings of International Workshop on CBSE*, held in conjunction with the ICSE.

Homes, B. (2012). Fundamentals of software testing, *John Wiley & Sons*.

Janicki, M., M. Katara, et al. (2012). "Obstacles and opportunities in deploying model-based GUI testing of mobile software: a survey." *Software Testing, Verification and Reliability 22(5): 313-341*.

Karam, M. R., S. M. Dascalu, et al. (2006). "Challenges and opportunities for improving code-based testing of graphical user interfaces." Journal of Computational Methods in Sciences and Engineering 6(5, 6 Supplement 2): 379-388.

Kepple, L. R. (1992). "A new paradigm for cross-platform automated GUI testing." The X Resource 3(1): 155-178.

Lewis, W. E. (2004). "Software testing and continuous quality improvement." *CRC press*.

Ley, M. (1993). "Digital Bibliographic Library Browser (DBLP) Computer Science Bibliography." Retrieved 2014, from http://dblp.uni-trier.de/.

Li, P., T. Huynh, et al. (2007). "A practical approach to testing GUI systems." *Empirical Software Engineering 12(4): 331-357*.

Mao, Y., F. Boqin, et al. (2006). "Important usage paths selection for GUI software testing." *Information and Technology Journal 5(4): 648-654*.

Mateo Navarro, P. L., D. Sevilla Ruiz, et al. (2009). "A proposal for automatic testing of GUIs based on annotated use cases." *Advances in Software Engineering vol. 2010*.

McMaster, S. and A. Memon (2008). "Call-stack coverage for gui test suite reduction." IEEE Transactions on Software Engineering 34(1): 99-115.

Memon, A. (2015). "GUITAR." Retrieved 2015, from http://sourceforge.net/projects/guitar/.

Memon, A., A. Nagarajan, et al. (2005). "Automating regression testing for evolving GUI software." *Journal of Software Maintenance and Evolution: Research and Practice 17(1): 27-64*.

Memon, A. M. (2006). "Employing user profiles to test a new version of a GUI component in its context of use." *Software Quality Journal 14(4): 359-377*.

Memon, A. M. (2007). "An event-flow model of GUI-based applications for testing." *Software Testing Verification and Reliability 17(3): 137-158*.

Memon, A. M. (2008). "Automatically repairing event sequence-based GUI test suites for regression testing." *ACM TOSEM 18(2): 4*.

Memon, A. M., M. E. Pollack, et al. (2001). "Hierarchical GUI test case generation using automated planning." *IEEE Transactions on Software Engineering 27(2): 144-155*.

Memon, A. M. and Q. Xie (2005). "Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software." *IEEE Transactions on Software Engineering 31(10): 884-896*.

Nguyen, B. N., B. Robbins, et al. (2014). "GUITAR: an innovative tool for automated testing of gui-driven software." *Automated Software Engineering 21(1): 65-105*.

Pham, R., H. Holzmann, et al. (2014). "Tailoring video recording to support efficient GUI testing and debugging." *Software Quality Journal 22(2): 273-292*.

Rubel, D. and P. Quitslund (2007). Automating GUI testing for Eclipse RCP applications, *Software Test and Performance*.

Takahashi, J. (2001). "An automated oracle for verifying GUI objects." *ACM SIGSOFT Software Engineering Notes 26(4): 83-88*.

Tsujino, Y. (2000). "A verification method for some GUI

dialogue properties." *Systems and Computers in Japan 31(14): 38-46*.

Xie, Q. and A. M. Memon (2007). "Designing and comparing automated test oracles for GUI-based software applications." *ACM TOSEM 16(1): 4*.

Xie, Q. and A. M. Memon (2008). "Using a pilot study to derive a GUI model for automated testing." *ACM TOSEM 18(2): 7*.

Yang, W., Z. Chen, et al. (2014). "GUI testing assisted by human knowledge: Random vs. functional." *Journal of Systems and Software 89: 76-86.*

Ye, M., B. Feng, et al. (2007). "Automated oracle based on multi-weighted neural networks for gui testing." *Information Technology Journal 6(3): 370-375*.

Yuan, X., M. B. Cohen, et al. (2011). "GUI interaction testing: Incorporating event context." *IEEE Transactions on Software Engineering 37(4): 559-574*.

Yuan, X. and A. M. Memon (2010). "Generating event sequence-based test cases using GUI runtime state feedback." *IEEE Transactions on Software Engineering 36(1): 81-95*.

Yuan, X. and A. M. Memon (2010). "Iterative execution-feedback model-directed GUI testing." *Information and Software Technology 52(5): 559-575*.