Information and Software Technology 55 (2013) 1679–1694

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Graphical user interface (GUI) testing: Systematic mapping and repository

Ishan Banerjee[a], Bao Nguyen[a], Vahid Garousi[b,c,*], Atif Memon[a]

[a] Department of Computer Science, University of Maryland, College Park, MD 20742, USA
[b] Electrical and Computer Engineering, University of Calgary, Calgary, Canada
[c] Informatics Institute, Middle East Technical University, Ankara, Turkey

A R T I C L E I N F O

Article history:
Received 30 August 2012
Received in revised form 11 March 2013
Accepted 11 March 2013
Available online 5 April 2013

Keywords:
Systematic mapping
GUI application
Testing
Paper repository
Bibliometrics

A B S T R A C T

Context: *GUI testing* is system testing of a software that has a graphical-user interface (GUI) front-end. Because system testing entails that the entire software system, including the user interface, be tested as a whole, during GUI testing, test cases—modeled as sequences of user input events—are developed and executed on the software by exercising the GUI's widgets (e.g., text boxes and clickable buttons). More than 230 articles have appeared in the area of GUI testing since 1991.
Objective: In this paper, we study this existing body of knowledge using a systematic mapping (SM).
Method: The SM is conducted using the guidelines proposed by Petersen et al. We pose three sets of research questions. We define selection and exclusion criteria. From the initial pool of 230 articles, published in years 1991–2011, our final pool consisted of 136 articles. We systematically develop a classification scheme and map the selected articles to this scheme.
Results: We present two types of results. First, we report the demographics and bibliometrics trends in this domain, including: top-cited articles, active researchers, top venues, and active countries in this research area. Moreover, we derive the trends, for instance, in terms of types of articles, sources of information to derive test cases, types of evaluations used in articles, etc. Our second major result is a publicly-accessible repository that contains all our mapping data. We plan to update this repository on a regular basis, making it a "live" resource for all researchers.
Conclusion: Our SM provides an overview of existing GUI testing approaches and helps spot areas in the field that require more attention from the research community. For example, much work is needed to connect academic model-based techniques with commercially available tools. To this end, studies are needed to compare the state-of-the-art in GUI testing in academic techniques and industrial tools.

Contents

* Corresponding author at: Electrical and Computer Engineering, University of Calgary, Calgary, Canada. Tel.: +1 403 210 5412.
E-mail addresses: ishan@cs.umd.edu (I. Banerjee), baonn@cs.umd.edu (B. Nguyen), vgarousi@ucalgary.ca (V. Garousi), atif@cs.umd.edu (A. Memon).

0950-5849/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.infsof.2013.03.004

1. Introduction

Whenever the number of *primary studies*—reported in *articles* (we use the term *article* to include research papers, book chapters, dissertations, theses, published experimental results, and published demonstrations of techniques)—in an area grows very large, it is useful to summarize the body of knowledge and to provide an overview using a secondary study [1]. A secondary study [2–5] aggregates and objectively synthesizes the outcomes of the primary studies. By "mapping the research landscape," a secondary study helps to identify sub-areas that need more primary studies. Because the synthesis needs to have some common basis for extracting attributes in the articles, a side effect of the secondary study is that it encourages researchers conducting and reporting primary studies to improve their reporting standard of such attributes, which may include metrics, tools, study subjects, limitations, etc.

In the field of Software Engineering (SE), a systematic mapping (SM) study is a well-accepted method to identify and categorize research literature [6,1]. An SM [2,7–12] study focuses on building classification schemes and the results show frequencies of articles for classifications within the scheme. These results become one of the outputs of the SM in the form of a database or *map* that can be a useful descriptive tool itself. An SM uses established searching protocols and has rigorous inclusion and exclusion criteria.

In this paper, we leverage the guidelines set by Petersen et al. [1] and Kitchenham and Charters [13] to create an SM for the area of *GUI testing*. We define the term GUI testing to mean that a GUI-based application, i.e., one that has a graphical-user interface (GUI) front-end, is tested solely by performing sequences of events (e.g., *"click on button"*, *"enter text"*, *"open menu"*) on GUI *widgets* (e.g., *"button"*, *"text-field"*, *"pull-down menu"*). In all but the most trivial GUI-based systems, the space of all possible event sequences that may be executed is extremely large, in principle infinite, e.g., consider the fact that a user of Microsoft Word can click on the *File* menu an unlimited number of times. All GUI testing techniques are in some sense *sampling* the input space, either manually [14,15] or automatically [16,17]. In the same vein, techniques that develop a GUI *test oracle* [18]—a mechanism that determines whether a GUI executed correctly for a test input—are based on sampling the output space; examining the entire output, pixel by pixel, is simply not practical [19,20]. Techniques for evaluating the adequacy of GUI test cases provide some metrics to quantify the test cases [21–23]. Techniques for regression testing focus on retesting the GUI software after modifications [24–26].

The above is just one possible classification of GUI testing techniques. The goal of our SM is to provide a much more comprehensive classification of articles that have appeared in the area since 1991 (our search revealed that the first paper on GUI testing appeared in 1991). Given that now there are regular events such as the International Workshop on TESTing Techniques & Experimentation Benchmarks for Event-Driven Software (TESTBEDS) [27–30] in the area, we expect this number to increase. We feel that this is an appropriate time to discuss trends in these articles and provide a synthesis of what researchers think are limitations of existing techniques and future directions in the area. We also want to encourage researchers who publish results of primary studies to improve their reporting standards, and include certain attributes in their articles to help conduct secondary studies. Considering that many computer users today use GUIs exclusively and have encountered GUI-related failures, research on GUIs and GUI testing is timely and relevant.

There have already been 2 smaller, preliminary secondary studies on GUI testing. Hellmann et al. [31] presented a literature review of test-driven development of user interfaces; it was based on a sample of 6 articles. Memon and Nguyen [32] presented a classification of 33 articles on model-based GUI test-case generation techniques. To the best of our knowledge, there are no other secondary studies in the area of GUI testing.

In our SM, we study a total of 230 articles. We formulate 3 sets of research questions pertaining to the research space of GUI testing, demographics of the studies and authors, and synthesis and interpretation of findings. We describe the mechanisms that we used to locate the articles and the set of criteria that we applied to exclude a number of articles; in all we classify 136 articles. Our most important findings suggest that there is an increase in the number of articles in the area; there has been lack of evaluation and validation, although this trend is changing; there is insufficient focus on mobile platforms; new techniques continue to be developed and evaluated; evaluation subjects are usually non trivial, mostly written in Java, and are often tested using automated model-based tools; and by far a large portion of the articles are from the US, followed by China.

We have published our SM as an online repository on Google Docs [33]. Our intention is to periodically update this repository, adding new GUI testing articles as and when they are published. In the future, we intend to allow authors of articles to update the repository so that it can become a "live" shared resource maintained by the wider GUI testing community.

The remainder of this paper is structured as follows. Section 2 presents background on GUI testing. Section 3 presents our goals and poses research questions. The approach that we used to select articles is presented in Section 4. Section 5 presents the process used for constructing the systematic map. Sections 6–8 present the results of the systematic mapping. Section 9 presents a discussion of results. Finally, Section 10 presents related work and Section 11 concludes with a summary of our findings and future work.

2. Background

A GUI takes events (mouse clicks, selections, typing in text-fields) as input from users, and then changes the state of its widgets. GUIs have become popular because of the advantages this "event-handler architecture" offers to both developers and users [34,35]. From the developer's point of view, the event handlers may be created and maintained fairly independently; hence, complex systems may be built using these loosely coupled pieces of code. From the user's point of view, GUIs offer many degrees of usage freedom, i.e., users may choose to perform a given task by inputting GUI events in many different ways in terms of their type, number and execution order.

Software testing is a popular QA technique employed during software development and deployment to help improve its quality [36,37]. During software testing, test cases are created and executed on the software under test. One way to test a GUI is to execute each event individually and observe its outcome, thereby testing each event handler in isolation [16]. However, the execution outcome of an event handler may depend on its internal state, the state of other entities (objects, event handlers) and the external environment. Its execution may lead to a change in its own state or that of other entities. Moreover, the outcome of an event's execution may vary based on the sequence of preceding events seen thus far. Consequently, in GUI testing, each event needs to be tested in different states. GUI testing therefore involves generating and executing sequences of events [38,35]. Most of the articles on test-case generation that we classify in our SM consider the event-driven nature of GUI test cases, although few mention it explicitly.

3. Goals, questions, and metrics

We use the Goal–Question–Metric (GQM) paradigm [39] to form the goals of this SM, raise meaningful research questions, and carefully identify the metrics that we collect from the primary studies and how we use them to create our maps. The goals of this study are:

G1: To classify the nature of articles in the area of GUI testing, whether new techniques are being developed, whether they are supported by tools, their weaknesses and strengths, and to highlight and summarize the challenges and lessons learned.
G2: To understand the various aspects of GUI testing (e.g., test-case generation, test coverage) that are being investigated by researchers.
G3: To study the nature of evaluation, if any, that is being conducted, the tools being used, and subject applications.
G4: To identify the most active researchers in this area and their affiliations, and identify the most influential articles in the area.
G5: To determine the recent trends and future research directions in this area.

Goals **G1**, **G2**, and **G3** are all related to understanding the trends in GUI testing *research and evaluation* being reported in articles. These goals lead to our first set of research questions. Note that as part of the research questions, we include the metrics (underlined) that we collect for the SM.

RQ 1.1: *What types of articles have appeared in the area?* For example, we expect some articles that present new techniques, others that evaluate and compare existing techniques.
RQ 1.2: *What test data generation approaches have been proposed?* For example, some test data may be obtained using manual approaches, others via automated approaches. We examine this question because of the central role that test data plays in software testing.
RQ 1.3: *What type of test oracles have been used?* A test oracle is a mechanism that determines whether a test case passed or failed. A test case that does not have a test oracle is of little value as it will never fail. We expect some test cases to use a manual test oracle, i.e., manual examination of the test output to determine its pass/fail status. Other test cases may use an automated test oracle, in which the comparison between expected and actual outputs is done automatically.
RQ 1.4: *What tools have been used/developed?* We expect that some techniques would have resulted in tools; some are based on existing tools. Here we want to identify the tools and some of their attributes, e.g., execution platform.
RQ 1.5: *What types of systems under test (SUT) have been used?* Most new techniques need to be evaluated using some software subjects or SUTs. We want to identify these SUTs, and characterize their attributes, e.g., platform (such as mobile, web), size in lines of code (LOC).
RQ 1.6: *What types of evaluation methods have been used?* We expect that some techniques would have been evaluated using the type and amount of code that they cover, others using the number of test cases they yield, and natural or seeded faults they detected.
RQ 1.7: *Is the evaluation mechanism automated or manual?* A new technique that can be evaluated using automated mechanisms (e.g., code coverage using code instrumentation) makes it easier to replicate experiments and conduct comparative studies. Widespread use of automatic mechanisms thus allows the research area to encourage experimentation.

To answer all the above questions, we carefully examine the articles, collect the relevant metrics, create classifications replying explicitly on the data and findings reported in the articles, and obtain frequencies when needed. All the metrics are objective, i.e., we do not offer any subjective opinions to answer any of these questions.

Goals **G4** and parts of **G1** and **G5** are concerned with understanding the *demographics and bibliometrics* of the articles and authors. These goals lead to our second set of research questions.

RQ 2.1: *What is the annual articles count?*
RQ 2.2: *What is the article count by venue type?* We expect the most popular venues to be conferences, workshops, and journals.
RQ 2.3: *What is the citation count by venue type?*
RQ 2.4: *What are the most influential articles in terms of citation count?*
RQ 2.5: *What are the venues with highest articles count?*
RQ 2.6: *What are the venues with highest citation count?*
RQ 2.7: *Who are the authors with the highest number of articles?*
RQ 2.8: *What are the author affiliations*, i.e., *do they belong to academia or industry?*
RQ 2.9: *Which countries have produced the most articles?*

Again, we observe that the above questions may be answered by collecting objective metrics from the articles.

Goals **G5** and parts of **G1** are concerned with the *recent trends, limitations, and future research directions* in the area of GUI testing; we attain these goals by studying recent articles, the weaknesses/strengths of the reported techniques, lessons learned, and future trends. More specifically, we pose our third set of research questions.

RQ 3.1: *What limitations have been reported?* For example, some techniques may not scale for large GUIs.
RQ 3.2: *What lessons learned are reported?*
RQ 3.3: *What are the trends in the area?* For example, new technologies may have prompted researchers to focus on developing new techniques to meet the needs of the technologies.
RQ 3.4: *What future research directions are being suggested?*

Due to the nature of the questions and to prevent our own bias, their answers are based on opinions of the original authors who conducted the primary studies.

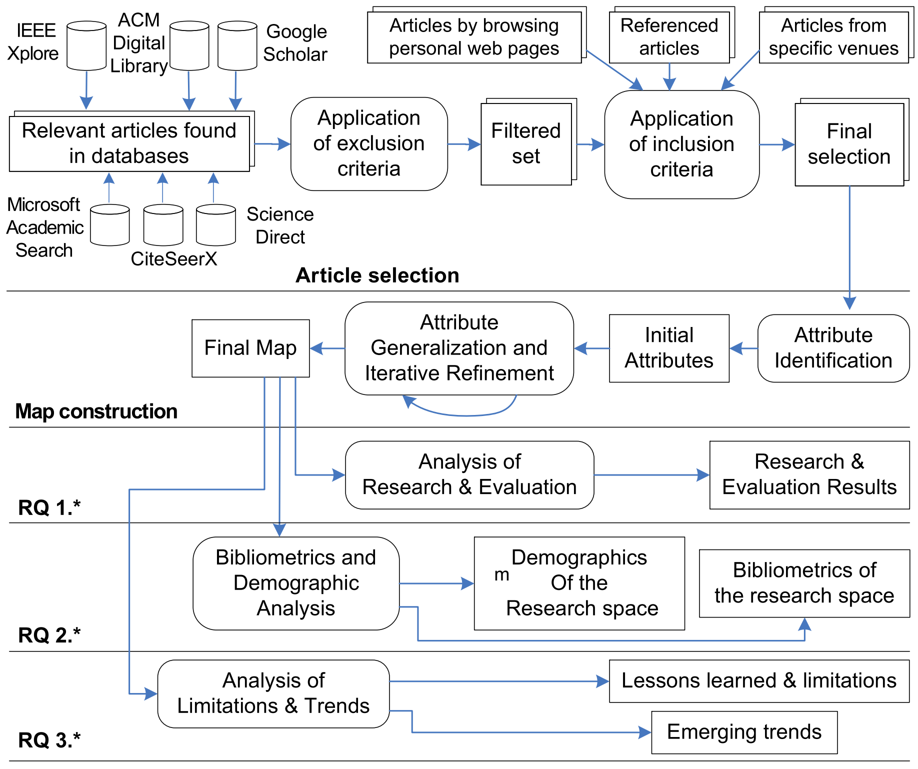

IEEE Xplore
ACM Digital Library
Google Scholar
Articles by browsing personal web pages
Referenced articles
Articles from specific venues
Relevant articles found in databases
Application of exclusion criteria
Filtered set
Application of inclusion criteria
Final selection
Microsoft Academic Search
CiteSeerX
Science Direct
Article selection
Final Map
Attribute Generalization and Iterative Refinement
Initial Attributes
Attribute Identification
Map construction
Analysis of Research & Evaluation
Research & Evaluation Results
RQ 1.*
Bibliometrics and Demographic Analysis
Demographics Of the Research space
Bibliometrics of the research space
RQ 2.*
Analysis of Limitations & Trends
Lessons learned & limitations
Emerging trends
RQ 3.*

Fig. 1. Protocol guiding this SM. The five distinct phases are **Article selection**, **Map construction**, **RQ 1.**∗, **RQ 2.**∗, **RQ 3.**∗.

Having identified the goals for this work, linking them to research questions, and identifying the metrics that we collect, we have set the stage for the SM. The remainder of this paper is based on the protocol that lies at the basis of this SM; it is outlined in Fig. 1. The figure describes the workflow, inputs and outputs of the SM. Each row in this figure is a distinct phase having clearly defined inputs and outputs. From this figure, the protocol distinguishes five such phases. They are described in Sections 4–8. More specifically, we describe the process of **Article Selection** in Section 4, **Map Construction** in Section 5, and address research questions **RQ 1.**∗ in Section 6, **RQ 2.**∗ in Section 7, and **RQ 3.**∗ in Section 8.

4. Article selection

As can be imagined, article selection is a critical step in any secondary study. Indeed, it lays the foundation for the synthesis of all of its results. Consequently, in any secondary study, article selection must be explained carefully so that the intended audience can interpret the results of the study keeping in mind the article selection process. In this work, the articles were selected using a three step process using guidelines presented in previous systematic mapping articles [40,13,1]: (1) *article identification*, done using digital libraries and search engines, (2) *definition and application of exclusion criteria*, which exclude articles that lie outside the scope of this study, and (3) *definition and application of inclusion criteria*, which target specific resources and venues that may have been missed by the digital libraries and search engines to hand-pick relevant articles. These steps are illustrated in the top part of Fig. 1. We now expand upon each step.

4.1. Step 1: Article identification

We started the process by conducting a keyword-based search to extract a list of articles from the following digital libraries and search engines: IEEE Xplore,[1] ACM Digital Library,[2] Google Scholar,[3] Microsoft Academic Search,[4] Science Direct,[5] and CiteSeerX.[6] The following keywords were used for searching: *GUI testing*, *graphical user interface testing*, *UI testing*, and *user interface testing*; we looked for these keywords in article titles and abstracts. This step yielded 198 articles forming the initial pool of articles.

4.2. Step 2: Exclusion criteria

In the second step of the process, the following set of exclusion criteria were defined to exclude articles from the above initial pool. **C1**: languages other than English, **C2**: not relevant to the topic, and **C3**: that did not appear in the published proceedings of a conference, symposium, or workshop, or did not appear in a journal or magazine.

These criteria were then applied by defining application procedures. It was fairly easy to apply criterion **C1** and **C3**. For criterion **C2**, a voting mechanism was used amongst us (the authors) to assess the relevance of articles to GUI testing. We focused on the inclusion of articles on functional GUI testing; and excluded articles on non-functional aspects of GUIs, such as stress testing GUI applications [41] and GUI usability testing [42]. Application of the above exclusion criteria resulted in a filtered set of 107 articles.

4.3. Step 3: Inclusion criteria

Because search engines may miss articles that may be relevant to our study, we supplemented our article set by manually

1 http://ieeexplore.ieee.org/.
2 http://dl.acm.org/.
3 http://scholar.google.com/.
4 http://academic.research.microsoft.com/.
5 http://www.sciencedirect.com/.
6 http://citeseer.ist.psu.edu.

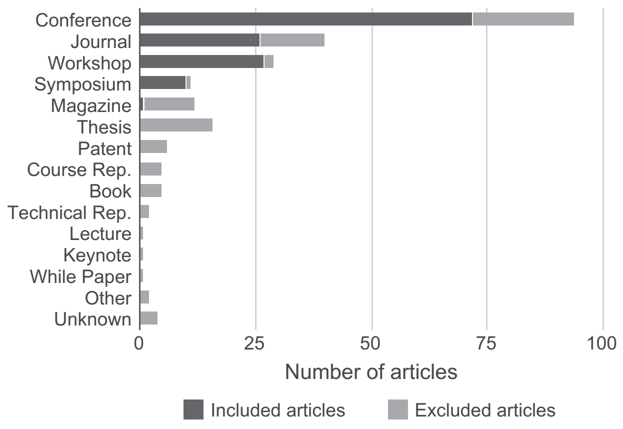


Fig. 2. Total articles studied = 230; final included = 136.

examining the following three sources: (1) web pages of active researchers, (2) bibliography sections of articles in our filtered pool, and (3) specific venues. These sources led to the definition of 3 corresponding inclusion criteria, application of which resulted in the final pool of articles containing 136 articles.

4.4. Our final article set

Fig. 2 shows the distribution of the 230 articles analyzed during this study. The dark shaded part of each horizontal bar shows the number that we finally included, forming a total of 136 articles. A few articles are classified as "Unknown" because, despite numerous attempts, we were unable to obtain their full-text version. In summary, we have included all articles presented at all venues that print their proceedings or make them available digitally.

5. Iterative development of the systematic map

As mentioned earlier, a map is the tool used for classification of the selected articles. Development of the map is a complex and time-consuming process. Indeed the map that we have made available in a publicly accessible repository [33] is one of the most important contributions of our work. Fortunately, because we use the GQM approach, we already have research questions and metrics; we use the metrics as a guide for map construction. For **RQ 1**.∗, we need to collect the metrics: "types of articles," "test data generation approaches," "type of test oracles," "tools," "types of SUT," "types of evaluation methods," and "evaluation mechanism." This list in fact forms a set of *attributes* for the articles. We define these attributes in this section and present the map structure. With this map (also called attribute framework [43]), the articles under study can be characterized in a comprehensive fashion.

The map was developed in an iterative manner. In the first iteration, all articles were analyzed and terms which appeared to be of interest or relevance for a particular aspect (e.g., 'subject under test', 'testing tool'), were itemized. This itemization task was performed by all of us. To reduce individual bias, we did not assume any prior knowledge of any attributes or keywords. The result after analyzing all articles was a large set of initial attributes. After the initial attributes were identified, they were generalized. This was achieved through a series of meetings. For example, under "test data generation approaches," the attributes 'finite-state machine (FSM)-based' method and 'UML-based' method were generalized to 'model-based' method.

Defining attributes for "types of articles" was quite complex. As one can imagine, there are innumerable ways of understanding the value of a research article. To make this understanding methodical, we defined two *facets*—specific ways of observing a subject—which helped us to systematically understand the contribution and research value of each article. The specific facets that we used, i.e., *contribution* and *research* were adapted from [1].

The resulting attributes for each facet were documented, yielding a map that lists the aspects, attributes within each aspect, and brief descriptions of each attribute. This map forms the basis for answering the research questions **RQ 1**.∗.

Similarly, for **RQ 2**.∗ we need the following metrics: "annual articles count," "article count by venue type," "citation count by venue type," "citation count," "citation count by venue," "venues with highest article counts," "authors with maximum articles," "author affiliations," and "countries." The first two metrics were obtained directly from our spreadsheet. The remaining metrics lead us to develop our second map. As before, the map lists the attributes and brief descriptions of each attribute. This map forms the basis for answering the research questions **RQ 2**.∗.

Finally, for **RQ 3**.∗, we need to collect the metrics: "limitations," "lessons learned," "trends," and "future research directions." This led us to develop our third map, which forms the basis for answering the research questions **RQ 3**.∗. The final map used in this research for all questions is shown in Fig. 3.

6. Mapping research and evaluation

We are now ready to start addressing our original research questions **RQ 1.1** through **RQ 1.7**.

RQ 1.1: *What types of articles have appeared in the area?* As discussed earlier in Section 5, we address this question using two facets, primarily taken from [1]. The contribution facet broadly categorizes the type of contributions(s) made by a given article, and can be one or more of the following types: test method, test tool, test model, metric, process, challenge, and empirical study. On the other hand, the *research facet*—solution proposal, validation, evaluation research, experience, philosophical and opinion articles—broadly categorizes the nature of research work presented in the article. Every article has been attributed with at least one category. Some articles have been placed in more than one category. For example, Belli [44] presents a testing technique based on FSMs. This article is placed under both 'test method' as well as 'test model' in contribution facet.

Fig. 4a shows the contribution facet for all the 136 articles. The *y*-axis enumerates the categories, and the *x*-axis shows the number of articles in each category. Most articles (90 articles) have contributed towards the development of new or improved testing techniques. Few articles have explored GUI testing metrics, or developed testing processes. Fig. 4c shows an annual distribution of the contribution facet. The *y*-axis enumerates the period 1991–2011, the *x*-axis enumerates the categories, the integer indicates the number of articles in each category for a year. During the period 1991–2000, most of the work focused on testing techniques. On the other hand, during 2001–2011, articles have contributed to various categories. This trend is likely owing to the rising interest in GUI testing in the research community.

Fig. 4b shows the research facet for all the 136 articles. Most articles propose solutions, conduct various types of experiments to validate techniques. There are very few philosophical or opinion articles. Fig. 4d shows an annual distribution of the research facet. The figure shows that there is an increasing number of articles in recent years, with most articles in solution proposal, validation and evaluation research facets. In the year 2011, the largest number of articles were on validation research, a promising development, showing that researchers are not only proposing novel techniques, but they are also supporting them with lab experiments.

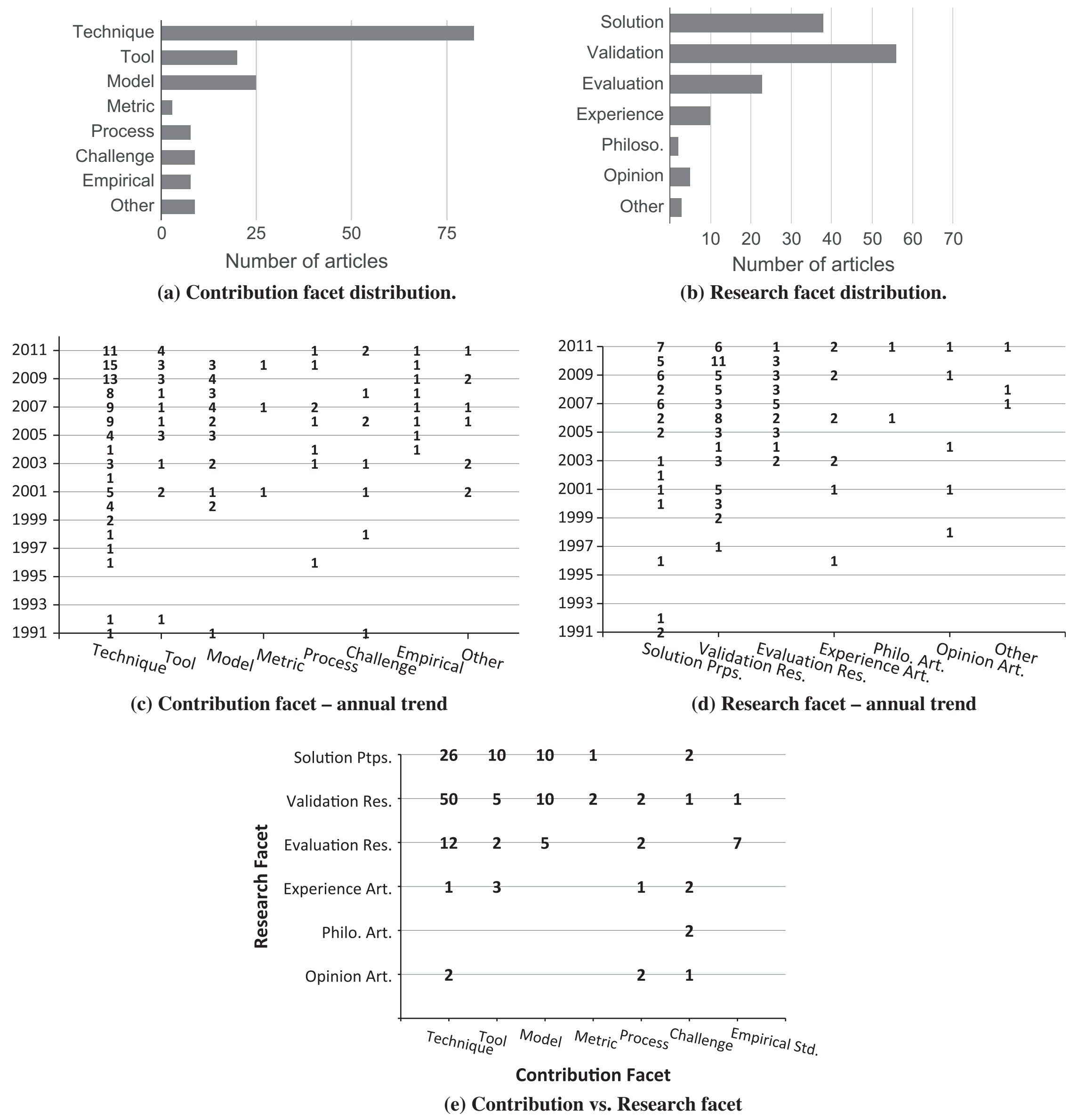
RQ 1.1: Type of article	*Contribution Facet*	Test method/technique (B)	Article describes new technique or improves upon an existing one
		Test tool (B)	Article focuses on testing tool and evaluates its applicability.
		Test model (B)	Article introduces new modeling technique or is based on use of model
		Metric (B)	Article describes new metric for evaluating testing techniques
		Process (B)	Article describes software testing process or life-cycle
		Challenge (B)	Article discusses challenges in certain areas of GUI testing
		Empirical study (B)	Article is an empirical study of technique
	Research Facet	Solution proposal (B)	New solution; applicability shown via example or line of argument
		Validation research (B)	Novel technique demonstrated in lab with experiment
		Evaluation research (B)	Comprehensive experimental evaluation of technique
		Experience article (B)	Personal experience with GUI testing of authors
		Philosophic article (B)	Sketch new way of looking at existing things.
		Opinion article (B)	Opinion of authors on the goodness of techniques.
RQ 1.2	Test data generation	Capture/replay (B)	Capture/replay was used to generate test cases
		Model based (B)	GUI model was used to generate test cases
		Model name (S)	Name of model used (if model-based)
		Random testing (B)	Test cases were generated randomly
RQ 1.3	Test oracle	State reference (B)	GUI state information was used as oracle
		Crash testing (B)	SUT crash was used to identify faults
		Formal specification (B)	Formal specification of SUT was used as oracle
		Manual verification (B)	Result of test case execution was manually verified
		Multiple oracles (B)	More than one oracle was used in same test run
RQ 1.4	Testing tools	Tool proposed (S)	Name of new tool introduced in an article
		Tool used (S)	Name of existing or third party tool used in an article
		Programming language (S)	Programming language used in developing the tool
RQ 1.5	System under test	Number of SUT(s) (N)	Number of SUT(s) used in the article
		Size (LOC) (N)	Number of lines of code in SUT
		Programming language (S)	Programming language of the SUT
		GUI technology (S)	GUI SDK or library used in SUT
		Small/large scale (S)	Qualitative assesment of the size of SUT
RQ 1.7	Evaluation Automation	Automated (B)	Automated test case execution was used in article
		Manual (B)	Manual test case execution was used in article
		None (B)	Test cases were not executed
RQ 2.*	Demographic Information	Authors (S)	Name of all contributing authors
		Authors' country (S)	Country from which author published the article
		Authors' affiliations (E)	Are the authors from academia, industry or a mix of both
		Venue (S)	Where it was published
		Year (N)	Year of publication
		Citation count (N)	Number of times this work has been cited, per year, as of 2011
RQ 3.*	Limitations & Future	Limitations (S)	Limitations noted by article authors
		Lessons learned (S)	Lessons learned
		Future research (S)	Future research directions

* B = Boolean, E = Enumerated, N = Numeric, S = String

Fig. 3. The final map produced by and used in this research.

To get a better understanding of the contribution and research focus of each article, we also visualize the relationship between the research and contribution facets in Fig. 4e The *y*-axis enumerates the research facet categories; the *x*-axis enumerates the contribution facet categories. The intersection of each pair of categories is an integer whose value corresponds to the number of articles at that point. Work on exploring new and improved techniques dominate with focus on validation research with 46 articles.

RQ 1.2: *What test data generation approaches have been proposed?* Of the 136 articles, 123 articles reported generation of test artifacts. The term test "artifacts" in this context denotes any type



(a) Contribution facet distribution.

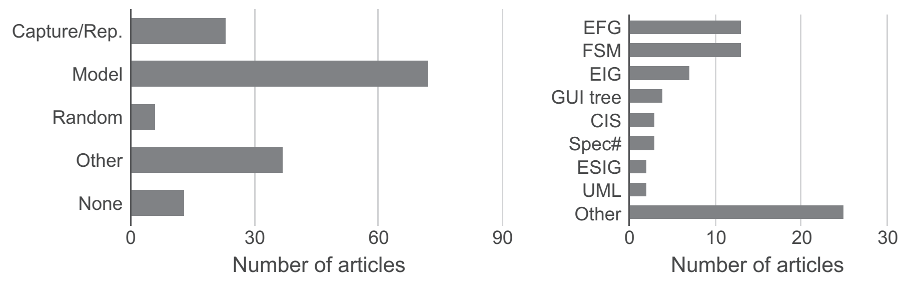
(b) Research facet distribution.

(c) Contribution facet – annual trend

(d) Research facet – annual trend

(e) Contribution vs. Research facet

Fig. 4. Data for **RQ 1.1**.



(a) Test generation methods

(b) Model based methods

Fig. 5. Data for **RQ 1.2**.

of artifacts generated and used for purpose of GUI testing, e.g., test cases, test input data, expected outputs and test oracle, test requirements, test harness, test code, etc. Fig. 5a shows the distribution of test data generation methods. The *x*-axis shows the number of articles for each method and the *y*-axis enumerates the methods.

By far, the most articles (72 articles) relied on a model for test generation. Fig. 5b shows the composition of these 72 articles.

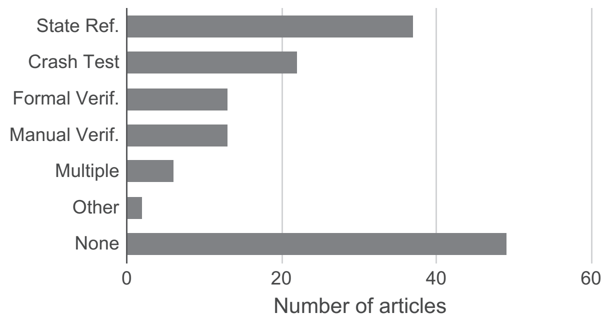


Fig. 6. Data for **RQ 1.3**: Oracle type.

The *x*-axis shows the number of articles using a model, *y*-axis enumerates the models. Models such as event flow graphs (EFG) and finite state machines (FSM) were most common. An EFG is a stateless directed graph that contains nodes that represent an event in the GUI and edges that represent a relationship between events. An edge from node e_x to node e_y means that the event represented by e_y may be performed immediately after the event represented by e_x. On the other hand, an FSM contains nodes that are explicit states and edges represent state transitions. For example, if the state represents the current GUI window (as in [45]), then the number of states is equal to the number of windows in the GUI. Events are represented by transitions because they may open/close windows, taking the FSM from one state to another. There are 25 articles which use less common models such as Probabilistic model [46] and Function trees [47].

Another popular approach (23 articles) was capture/replay (e.g., Ariss et al. [14]) in which a human tester interacts with the SUT, performing sequences of events. During the interaction, a capture/replay tool records/captures all the events performed and saves them in a database. A replay part of the tool can then automatically perform the recorded interactions, thereby mimicking the human tester.

The remaining 37 articles use less popular methods such as symbolic execution [48], formal methods [49], AI planning [16], and statistical analysis [50].

RQ 1.3: *What type of test oracles have been used?* We remind the reader that a test oracle is a mechanism that determines if a test case passed or failed. As Fig. 6 shows, state reference (37 articles) is the commonly used oracle. In this method the state of the GUI is extracted while the SUT is executing, and is stored. At a later time, this state may be compared with another execution instance for verification [20,19]. SUT crash testing is another popular oracle (22 articles). In this method, if the SUT crashes during the execution of a test case, then the test case is marked as failed, i.e., the 'crash' state of the SUT is interpreted as an oracle [22,51]. Formal verification methods (13 articles) use a model or specification to verify the correctness of the output of a test case [52,49]. Manual verification (13 articles) is also used. In this method a human tester is involved in verifying the result of executing a test case [17,53].

We observed that a large number of articles (49 articles) did not use a test oracle. Of these, 13 articles are experience, philosophical or opinion articles and do not require a test oracle for evaluation. The remaining 36 articles are solution proposal, validation or evaluation but do not use/propose a test oracle (e.g., [48,24]).

RQ 1.4: *What tools have been used/developed?* Testing of GUI based applications typically require the use of tools. A *tool*, for the purpose of this paper, is a set of well defined, packaged, distributable software artifacts which is used by a researcher to evaluate or demonstrate a technique. Test scripts, algorithm implementations and other software components used for conducting experiments, which were not named or made public, have not been considered as tools.

A tool is considered as a *new tool* if it has been developed specifically for use for the first time in an article. A tool is considered as an *existing tool* if it has been developed in a previous work or has been developed by a third party—commercially available, open source, etc.

Fig. 7a shows the composition of new and existing tools used for all 136 articles. It can be seen that 32 articles (23.52%) introduced a new tool only, 48 articles (35.29%) used an existing tool only, 29 articles (21.32%) used both new and existing tools, whereas 27 articles (19.85%) did not use a clearly defined tool. From this figure it can be seen that most articles (109 articles) used one or more tools. Certain articles, such as experience, philosophical and opinion articles, for example by Robinson et al. [54], did not require a tool.

From the 109 articles that used a tool, a total of 112 tools were identified. Note that a tool may have been used in more than one article. Similarly, an article may have used more than one tool. Fig. 7b shows the ten most popular tools and their usage count. The *x*-axis shows the number of articles where the tool was used, *y*-axis enumerates the 10 most popular tools. GUITAR [55], which ranks highest, has been used in 22 articles. 91 tools were used in only 1 article, 15 tools were used in 2 articles and so forth. Only 1 tool, GUITAR [55], was used in 22 articles.

New GUI testing tools were described in 61 articles. Fig. 7c shows the distribution of programming languages in which the tools were developed, which also usually implies the programming languages that it supports in terms of the SUT. The *x*-axis shows the number of articles in which a new tool was developed in a particular language, *y*-axis enumerates the languages. From the figure, Java is by far the most popular choice with 23 articles.

RQ 1.5: *What types of systems under test (SUT) have been used?* Of the 136 articles, 118 reported the use of one or more SUT. Note that an SUT may have been used in different articles, conversely, more than one SUT may have been used in an article. Fig. 8a shows the number of SUTs that were used in each article. This figure helps us understand how many SUTs are typically used by researchers to evaluate their techniques. The *x*-axis enumerates the SUT count from 1 to 6 and ⩾7. The *y*-axis shows the number of articles using

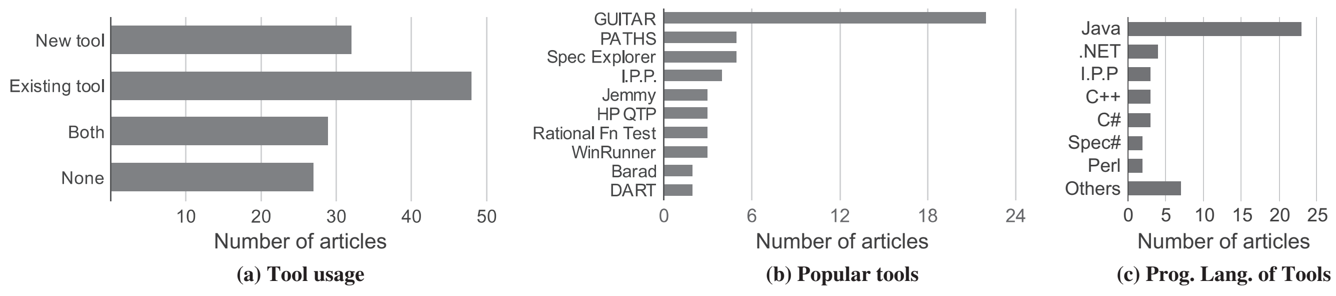


Fig. 7. Data for **RQ 1.4**.

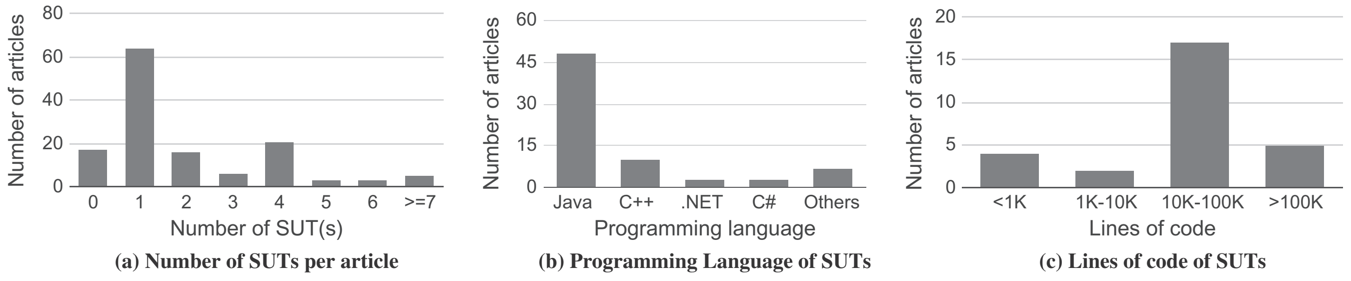


Fig. 8. Data for **RQ 1.5**.

a given number of SUTs. From the figure, it can be seen that out of 136 articles, 118 used one or more SUTs. Only 1 SUT was used in 64 articles, while a small number of articles (5) [56–59,47] used 7 or more SUTs. A total of 18 articles (e.g., [60]) did not use any SUT.

Fig. 8b shows the programming language of SUTs reported by 71 articles. The *x*-axis enumerates the common languages, *y*-axis shows the number of articles for each language. We see that Java applications are by far the most common SUT with 48 articles using Java based SUT(s) [35,61,62]. C/C++ [54,63] and .NET [64,65] based SUTs have been used in 16 articles. The remaining 7 SUTs are based on MATLAB [66], Visual Basic [67] and Objective C [68].

SUTs also differed in their underlying development technology. For example, some SUTs were developed using Java's Abstract Window Toolkit (AWT), while others were developed using the Swing technology. Classifying their development technology helps understand the experimental environment that are the focus of new GUI testing techniques. We found that Java Swing is by far the most common technology, with 25 out of 36 articles using an SUT based on Swing. This is consistent with a large number of SUTs being based on Java.

SUTs used for the evaluation of techniques proposed in the articles varied in size, in terms of lines-of-code (LOC)—some less than 1000 lines while some more than 100,000 lines. Only 28 articles out of 136 reported this information. If more than one SUT was used in a given article and if their LOC sizes were reported, we calculated the cumulative value of LOC measures. Fig. 8c shows the cumulative LOC of SUTs used in each article. The *x*-axis enumerates ranges of LOC, *y*-axis shows the number of articles in each range. Most articles used SUTs in the range 10,000–100,000 (17 articles). Only 5 articles [69,64,70,25,71] used SUTs with LOC totaling more than 100,000 lines.

The SUTs were also classified as *large-scale* or *small-scale*. This classification helps us understand if some articles used small or *toy* SUTs. SUTs such as commercially available software—Microsoft WordPad [72], physical hardware such as vending machines [73], mobile phones [74] and open source systems [25] have been classified as large-scale systems. SUTs such as a set of GUI windows [75], a set of web pages [60], small applications developed specifically for demonstration [76,77] have been classified as small-scale SUTs. Of the 118 articles which used one or more SUTs, 89 articles (75.42%) used a large-scale SUT.

RQ 1.6: *What types of evaluation methods have been used?* Many articles studied in this SM focused on the development of new GUI testing techniques. The techniques developed in the article were evaluated by executing test cases on an SUT. Different methods and metrics were applied to determine the effectiveness of the testing technique.

A total of 119 articles reported one or more evaluation methods. Fig. 9a shows the distribution of evaluation methods. The *x*-axis shows the count of articles in each method, eleven evaluation methods are enumerated on the *y*-axis. For example, 47 articles

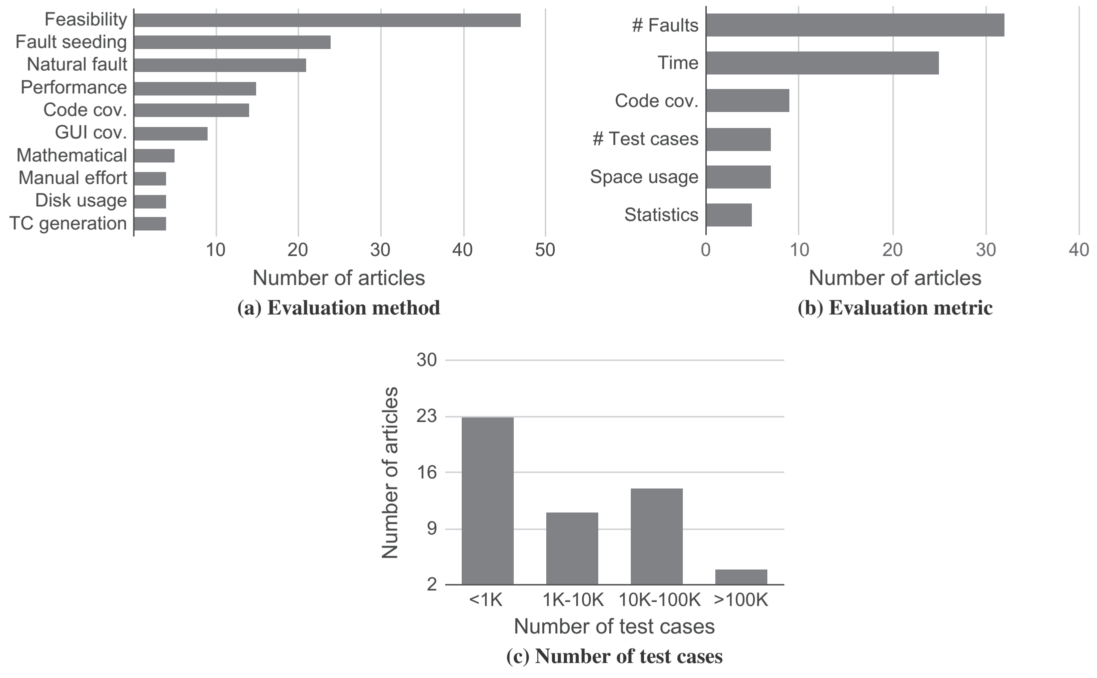


Fig. 9. Data for **RQ 1.6**.

demonstrated the feasibility of the technique using a simple example. Fig. 9b shows the metrics used in the evaluation. The *x*-axis shows the number of articles for each evaluation metric, *y*-axis enumerates evaluation metrics. Out of 136 articles, 75 articles specified an evaluation metric. Of these, the *number of faults detected* was the common metric (32 articles).

The number of generated test cases were reported and used in 52 of the 136 articles for the evaluation process. Fig. 9c shows the number of test cases used. The *x*-axis enumerates ranges of test case counts, *y*-axis shows the number of articles in each range. Most articles used less than 1000 test cases. Four articles used more than 100,000 test cases [78–80,35].

RQ 1.7: *Is the evaluation mechanism automated or manual?* Of the 136 articles, 86 articles reported execution of test cases for evaluation, of which 72 reported automated test case execution, 11 articles reported manual test case execution while 3 articles [81–83] reported both automated and manual test case execution.

7. Mapping demographics

We now address the **RQ 2.**∗ research questions set, which is concerned with understanding the demographics of the articles and authors.

RQ 2.1: *What is the annual articles count?* The number of articles published each year were counted. The trend of publication from 1991 to 2011 is shown in Fig. 10. An increasing trend in publication over years is observed. The two earliest articles in the pool were published in 1991 by Yip and Robson [84,85]. The total article counts of GUI testing articles in 2010 and 2011 was 19.

RQ 2.2: *What is the articles count by venue?* We classify the articles by venue type—conference, journal, workshop, symposium or magazine. Fig. 12 shows that the number of conference articles (72) are more than the articles in the other four categories combined (64).

RQ 2.3: *What is the citation count by venue type?* The number of citations for each article was extracted and aggregated for each venue type. Fig. 12 shows the number of citations from different venue types. Conferences articles have received the highest citations at 1544.

RQ 2.4: *What are the most influential articles in terms of citation count?* This research question analyzes the relationship between the citations for each article and its year of publication. Fig. 11 shows this data. The *x*-axis is the year of publication, and the *y*-axis is the number of citations. Each point in the figure represents an article.

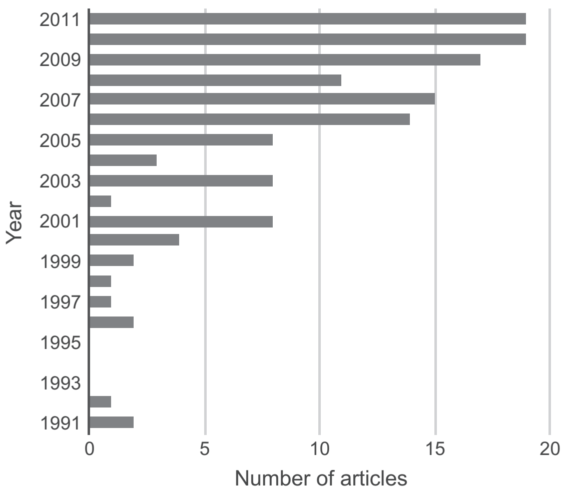


Fig. 10. RQ 2.1: Annual counts.

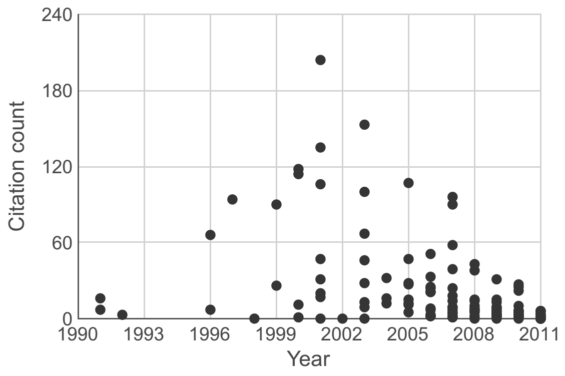


Fig. 11. RQ 2.4: Citations vs. year.

The points for the recent articles (from 2006 to 2011) are closer to each other, denoting that most of the recent articles have received relatively same number of citations, due to short time span as it takes time for a (good) article to have an impact in the area and be cited. The three earliest articles (two in 1991 and one in 1992) have received relatively low citations. The article with the highest number of citations is a 2001 IEEE TSE article by Memon et al. titled *'Hierarchical GUI Test Case Generation Using Automated Planning'* [16] and has received 204 citations.

RQ 2.5: *What were the venues with highest articles count?* Fig. 13 shows a count of articles from the top twenty venues, which contributed 80 articles. The annual International Workshop on TESTing Techniques & Experimentation Benchmarks for Event-Driven Software (TESTBEDS) is a relatively new venue, started in 2009. Since the venue has the specific focus on testing GUI and event-driven software, it has published the largest number, 16, of articles during 2009–2011. The International Conference on Software Maintenance (ICSM) with 8 and IEEE Transactions on Software Engineering (TSE) with 6 articles follow.

RQ 2.6: *What were the venues with highest citation count?* Fig. 14 shows that the top three cited venues are (1) IEEE TSE, (2) ACM SIGSOFT Symposium on the Foundations of Software (FSE) (3) International Symposium on Software Reliability Engineering (ISSRE). Some venues such as FSE did not publish many GUI testing articles (3). However, those articles have received a large number of citations (349). The correlation between the number of articles in each venue versus the total number of citations to those articles was 0.46 (thus, not strong).

RQ 2.7: *Who are the authors with maximum articles?* As Fig. 15 shows, Atif Memon (University of Maryland) stands first with 32 articles. The second and third highest ranking authors are Qing Xie (Accenture Tech Labs) and Mary Lou Soffa (University of Virginia) with 13 and 7 articles, respectively.

RQ 2.8: *What are the author affiliations, i.e., do they belong to academia or industry?* We classify the articles as coming from one of the following three categories based on the authors' affiliations: *academia*, *industry*, and *collaboration* (for articles whose authors come from both academia and industry). 73.52%, 13.23%, and 13.23% of the articles have been published by academics only, by industrial practitioners only, and with a collaboration between academic and industrial practitioners, respectively. The trend in each category over the years were tracked to see how many articles were written by academics or practitioners in different years. The results are shown in Fig. 16. There is a steady rise in the number of articles published from academia and industry in recent years. Also the number of collaborative articles between academics and practitioners has been on the rise.

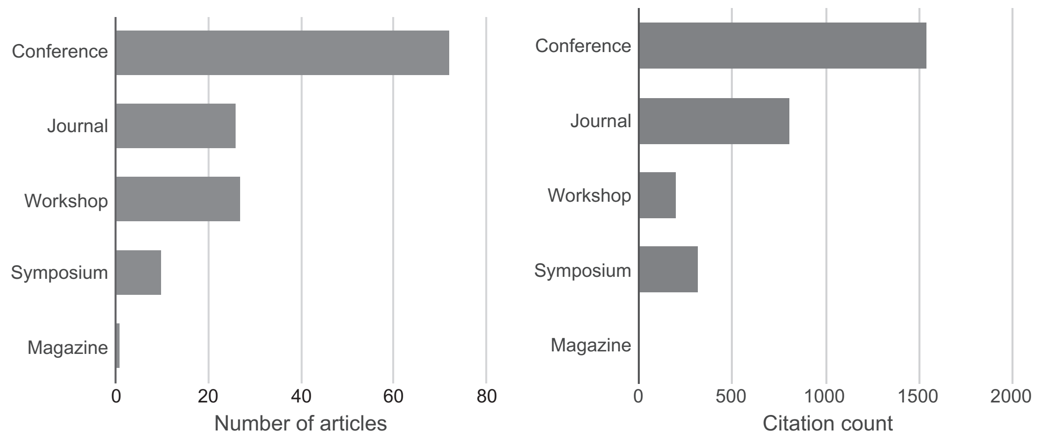


Fig. 12. **RQ 2.2** and **2.3**: Venue types.

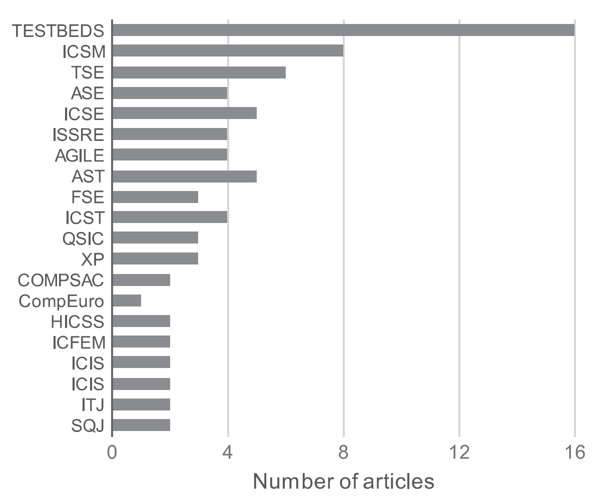


Fig. 13. Data for **RQ 2.5:** Top 20 venues.

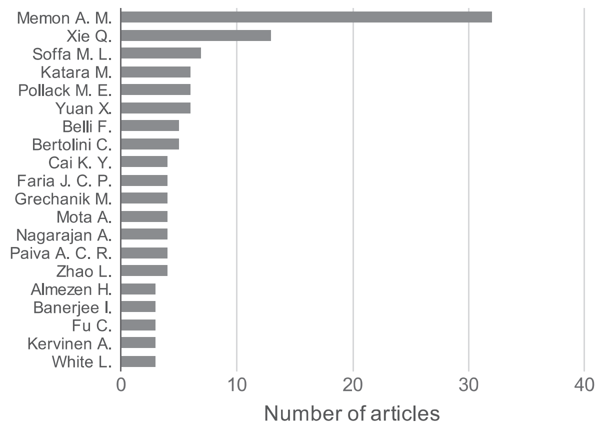


Fig. 15. Data for **RQ 2.7**: Top 20 authors.

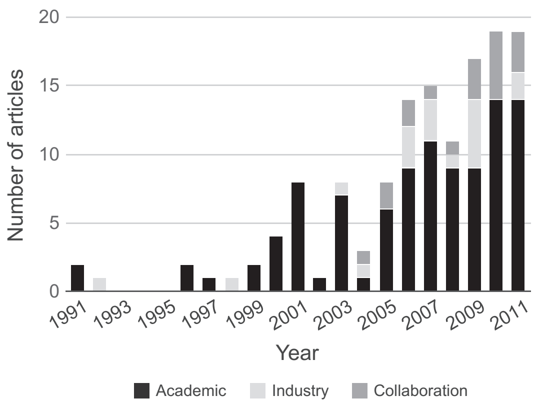


Fig. 16. Data for **RQ 2.8**: Author affiliation trend.

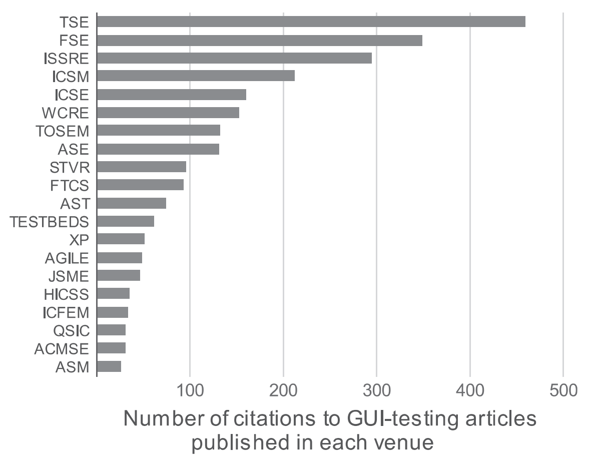


Fig. 14. Data for **RQ 2.6**: Venues most cited.

RQ 2.9: *Which countries have produced the most articles?* To rank countries based on number of articles published, the country of the residence of the authors was extracted. If a article had several authors from several countries, one credit for each country was assigned.

The results are shown in Fig. 17. The American researches have authored or co-authored 51.47% (70 of the 136) articles in the pool. Authors from China and Germany (with 12 and 9 articles, respectively) stand in the second and third ranks. Only 20 countries of the world have contributed to the GUI testing body of knowledge. International collaboration among the GUI testing researchers is quite under-developed as only 7 of the 136 articles were collaborations across two or more countries. Most of the remaining articles were written by researchers from one country.

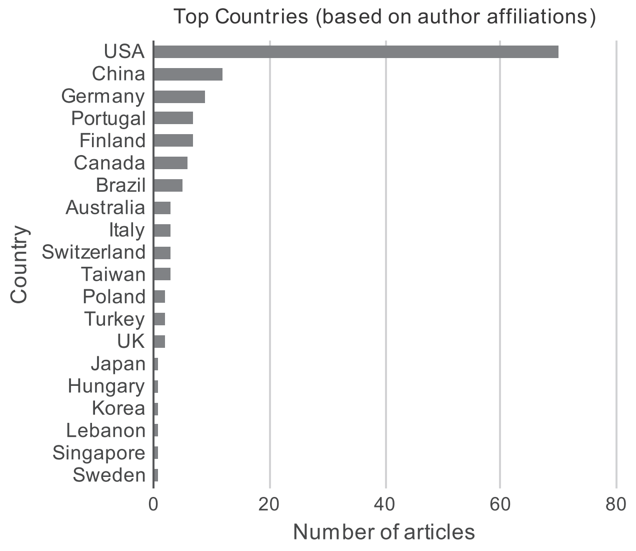


Fig. 17. Data for **RQ 2.9**: Top author countries.

8. Mapping limitations and future directions

The research questions **RQ 3.**∗ are addressed by classifying the *reported* limitations and future directions from the articles.

RQ 3.1: *What limitations have been reported?* Many of the articles explicitly stated limitations of the work. The limitations were broadly categorized as follows:

- *Algorithm:* The techniques or algorithms presented has known limitations—for example, an algorithm might not handle loops in the source code well [77].
- *Applicability:* Limitations on usability under different environments—for example, a tool or algorithm may be specific to AWT based applications [22].
- *Manual:* Manual steps are used in the experiments which may limit the usability and scalability of the method. Manual steps may also affect the quality of the experiment or technique—for example, manual effort may be required to maintain a model [86].
- *Oracle:* The oracle used for experiments may be limited in its capabilities at detecting all faults—for example, an oracle might be limited to detecting SUT crashes or exceptions [46], as opposed to comparing GUI states.
- *Fault:* Limitations on the ability to detect all kinds of faults or may detect false defects—for example, a tool might not handle unexpected conditions well [68].
- *Scalability:* The approach does not scale well to large GUIs—for example, time taken to execute the algorithm may increase exponentially with GUI size [87].
- *Tool:* There is some known tool limitation or obvious missing features in the tools used or proposed—for example, a tool may handle only certain types of GUI events [88].
- *Validity:* Experimental results are subject to internal or external validity [51,89].

Note that the authors of this SM did not judge the limitation themselves, but rather, we extracted the limitations when they were explicitly mentioned in each primary study. Out of the 136 articles, 45 articles reported one or more limitation of the research work. The extracted information is shown in Fig. 18a. This figure helps us understand the kind of limitations of the research work that were noted by the authors. The *x*-axis shows the number of articles in each category, the *y*-axis enumerates each category. The most common limitation is *validity*.

RQ 3.2: *What lessons learned are reported?* Only a small number of authors explicitly reported the lessons learned from their studies. Lesson learned were reported in only 11.76% (16/136) of all the articles. Lessons learned varied from author to author. They largely depend on individual research and study context. Hence, we conducted a qualitative analysis, instead of a quantitative analysis. It is important to note that they should be interpreted within the context of the studies.

Depending on the proposed testing techniques, the research lessons particularly associated with these techniques were reported. For example, in some cases, the authors who focus on model based testing where the model is created by hand, noted that in their approaches, a large amount effort would be spent on model creation [15]. In some other cases, the authors who used automated reverse engineered model based techniques, concluded that most of the tester's effort would be spent on test maintenance since the model is automatically created [20,90]. The model in those techniques can be obtained at a low cost.

Similarly, the experimentation environment has also influenced the authors' suggestions. Some authors with limited computation resources suggested that more research effort should be spent on test selection [91], test prioritization [17] and test refactoring [92] to reduce the number of test cases to execute. However, some

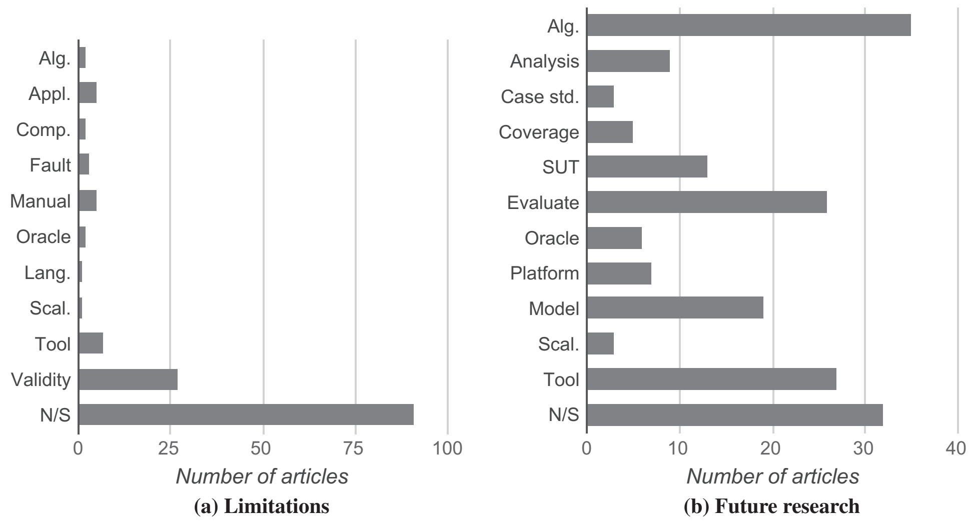


Fig. 18. Data for **RQ 3.**∗.

other authors with rich computation resources suggested that future research should focus on large scale studies [93].

RQ 3.3: *What are the trends in the area?* A widespread use of Java based SUTs and tools appears common. A notable development is the emergence of GUI testing work on mobile platforms during the years 2008–2011–8 article [88,94,46,95,89,96,73,97]), compared to only 1 article in the period 1991–2007 [74].

Another notable trend is a shift from unit script testing to large-scale automated system testing. Several large scale empirical studies have been conducted [51,69,38] thanks to the availability of automation tools and inexpensive computation resources.

RQ 3.4: *What future research directions are being suggested?* GUI testing is a relatively new research area in software engineering. Most of the articles provided guidance for continuing research, which may be broadly classified into the following categories:

- *Algorithmic:* Extend existing algorithms or develop new ones—for example, extend the algorithm to handle potentially large number of execution paths [98].
- *Analysis:* Further investigation based on results from the given study—for example, investigate interaction of different GUI components with *complete interaction sequence* (CIS) [17].
- *Coverage:* Coverage-based techniques presented in the article can be further improved or evaluated. The coverage technique may be applicable for either code, GUI or model coverage—for example, the study reported in [23] has developed a new coverage criterion.
- *Evaluate:* Evaluate of the proposed methods, and techniques further, extend investigation based on existing results—for example, conduct more controlled experiments [46].
- *Platform:* Extend the implementation for other platforms, e.g., web and mobile [87].
- *Model:* Improve or analyze the model presented in the article—for example, automatic generation of a model [99].
- *Scalability:* Scale the proposed algorithms to larger systems, reduce computation cost—for example, scaling the algorithm to handle larger GUIs while improving execution performance [100].
- *SUT:* Evaluate the proposed techniques with a more SUTs—for example, use complex SUTs for evaluation [101].
- *Tool:* Extend or add new capability or features to tools discusses in the article—for example, improve a tool to support better pattern matching and have better recovery from errors [102].

The future directions of research stated in the articles were extracted. Fig. 18b shows this data. This figure helps us understand what guidance has been provided by researchers. Although this data contains *future* directions dating back to the year 1991, it helps us understand the thoughts of researchers during this period and what they perceived as missing pieces at the time their work was performed and published.

In Fig. 18b the *x*-axis shows the number of articles in each category, the *y*-axis enumerates each category. It can be seen that improving algorithms (35 articles) as been perceived as the area requiring them most work. Improving and developing better GUI testing tools has also been perceived as an area requiring further work (27 articles).

9. Discussion

The systematic mapping presented in this paper documents certain aspects of GUI testing. In this section we present an objective summary of trends in GUI testing.

From the data collected, it can be seen that model-based GUI testing techniques have attracted the most attention in the research community. However, industrial tools such as JFCUnit,[7] Android's Monkey, Quick Test Pro,[8] Selenium[9] are not model based. There are no articles comparing the GUI testing techniques, methods and practices prevalent in the industry with those being developed in the research community. There has also been a general lack of collaboration between practitioners and researchers (see Fig. 16), although with exceptions in recent years.

In keeping with recent trends in computing technology, there has been a recent presence of GUI testing articles on mobile and web based platforms.

A large number of articles (109 out of 136) used GUI testing tools. As many as 112 tools were used. This indicates that while there has been a lot of effort in developing tools for GUI testing activities, there has not been much standardization of GUI testing tools. Different researchers develop their own custom tools for a specific research purpose. These tools are typically not usable by other researchers because they are not widely applicable. Also, the tools are often not maintained, debugged and developed over multiple years to be usable by other researchers.

From the data collected, most articles discuss new GUI testing techniques and tools. There has been a general lack of articles presenting opinion of researchers about the state-of-the-art techniques or providing guidance about possible future development and research directions.

10. Related work

10.1. Secondary studies in software testing

There have been 14 reported secondary studies in different areas of software testing, 2 related to GUI testing. We list these studies in Table 1 along with some of their attributes. For example, the "number of articles" column (No.) shows that the number of primary studies analyzed in each study varied from 6 (in [31]) to 264 (in [5]), giving some idea of the comprehensiveness of the studies.

In [31] a study of the interaction between agile test driven development methods and usability evaluation of user interfaces is presented. The article discusses the development of a low-fidelity prototype user-interfaces which are refined based on feedback from users.

Of particular interest to us are the SMs and structured literature reviews (SLRs). An SLR analyzes primary studies, reviews them in depth and describes their methodology and results. SLRs are typically of greater depth than SMs. Often, SLRs include an SM as a part of the study. Typically SMs and SLRs formally describe their search protocol and inclusion/exclusion criteria. We note that SMs and SLRs have recently started appearing in the area of software testing. There are four SMs: product lines testing [8], SOA testing [11], requirements specification and testing [7] and non-functional search-based software testing [2]. There are two SLRs—search-based non-functional testing [3] and search-based test-case generation [103].

The remaining eight studies are "surveys", "taxonomies", "literature reviews", and "analysis and survey", terms used by the authors themselves to describe their studies.

10.2. Online article repositories in SE

Authors of a few recent secondary studies have developed online repositories to supplement the study. This is a large

[7] http://sourceforge.net/projects/jfcunit/.
[8] www.hp.com/QuickTestPro.
[9] http://seleniumhq.org/.

Table 1
14 Secondary studies in software testing.

Type	Secondary study area	No.	Year	Refs.
SM	Non-functional search-based soft. testing	35	2008	[2]
	SOA testing	33	2011	[11]
	Requirements specification and testing	35	2011	[7]
	Product lines testing	45	2011	[8]
SLR	Search-based non-functional testing	35	2009	[3]
	Search-based test-case generation	68	2010	[103]
Survey	Object oriented testing	140	1996	[4]
	Testing techniques experiments	36	2004	[104]
	Search-based test data generation	73	2004	[105]
	Combinatorial testing	30	2005	[106]
	Symbolic execution for software testing	70	2009	[107]
Taxonomy	Model-based GUI testing	33	2010	[32]
Lit rev.	Test-driven development of user interfaces	6	2010	[31]
Analysis/survey	Mutation testing	264	2011	[5]

undertaking as even after the study is published, these repositories are updated regularly, typically every 6 months to a year. Maintaining and sharing such repositories provides many benefits to the broader community. For example, they are valuable resources for new researchers (e.g., PhD students) and for other researchers aiming to do additional secondary studies.

For example, Mark Harman and his team have developed and shared two online repositories, one in the area of mutation testing [5], and another in the area of search-based software engineering (SBSE) [108,109]. The latter repository is quite comprehensive and has 1014 articles as of 2011, a large portion of which are in search-based testing.

11. Conclusions

This SM is the most comprehensive mapping of articles in the area of *GUI Testing*. A total of 230 articles, from the years 1991–2011, were collected and studied, from which 136 articles were included in the SM.

Because of our own contributions to the field of GUI testing, we feel that we are reasonably qualified to provide an informed assessment of the survey results. First, we note that although there has been increased collaboration between academia and industry, no study has yet compared the state-of-the-art in GUI testing between academic and industrial tools and techniques. Such a study is sorely needed. Second, although there is a large focus on model-based testing with models such as FSM, EFG and UML in the literature, none of the commercially available tools are model based. This might signal a disconnect between researchers and practitioners. Some articles have started to bridge this gap [54,101] but much work is needed. Finally, an important result of this SM is that not all articles include information that is sought for secondary studies. We recommend that researchers working on GUI testing consider providing information in their articles using our maps as guides. Because of the increase in the number of articles on GUI testing, this is an ideal time to start supplying this information to support secondary studies. Our own effort in support of this endeavor is that we will continue to maintain an online repository [33] of GUI testing articles. We intend to continue analyzing the repository to create a systematic literature review (SLR).

Acknowledgements

The initial data collection stage of this work was started in a graduate course offered by Vahid Garousi in year 2010, in which the following students made some initial contributions: Roshanak Farhoodi, Shahnewaz A. Jolly, Rina Rao, Aida Shirvani, and Christian Wiederseiner. Their efforts are hereby acknowledged. Vahid Garousi was supported by the Discovery Grant 341511-07 from the Natural Sciences and Engineering Research Council of Canada and by the Visiting Scientist Fellowship Program (#2221) of the Scientific and Technological Research Council of Turkey (TÜBİTAK). The US authors were was partially supported by the US National Science Foundation (NSF) under NSF Grants CCF-0447864, CNS-0855055, and CNS-1205501.

References

[1] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: 12th International Conference on Evaluation and Assessment in Software Engineering, vol. 17, issue 1, 2007, pp. 1–10.
[2] W. Afzal, R. Torkar, R. Feldt. A systematic mapping study on non-functional search-based software testing, in: 20th International Conference on Software Engineering and Knowledge Engineering (SEKE 2008), 2008.
[3] W. Afzal, R. Torkar, R. Feldt, A systematic review of search-based testing for non-functional system properties, Information and Software Technology 51 (2009) 957–976.
[4] R.V. Binder, Testing object-oriented software: a survey, in: Proceedings of the Tools-23: Technology of Object-Oriented Languages and Systems, 1997, pp. 374–.
[5] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Transactions on Software Engineering 2008 (2010) 1–32.
[6] D. Budgen, M. Turner, P. Brereton, B. Kitchenham, Using mapping studies in software engineering, in: Proceedings of PPIG 2008, Lancaster University, 2008, pp. 195–204.
[7] Z.A. Barmi, A.H. Ebrahimi, R. Feldt, Alignment of requirements specification and testing: a systematic mapping study, in: Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, ICSTW '11, 2011, pp. 476–485.
[8] P.A. da Mota Silveira Neto, I.d. Carmo Machado, J.D. McGregor, E.S. de Almeida, S.R. de Lemos Meira, A systematic mapping study of software product lines testing, Information and Software Technology 53 (2011) 407–423.
[9] A. Fernandez, E. Insfran, S. Abrah£o, Usability evaluation methods for the web: a systematic mapping study, Information and Software Technology 53 (8) (2011) 789–817.
[10] B.A. Kitchenham, D. Budgen, O.P. Brereton, Using mapping studies as the basis for further research ¢ a participant–observer case study, Information and Software Technology 53 (6) (2011) 638–651.
[11] M. Palacios, J. García-Fanjul, J. Tuya, Testing in service oriented architectures with dynamic binding: a mapping study, Information and Software Technology 53 (2011) 171–189.
[12] J. Portillo-Rodriguez, A. Vizcaino, M. Piattini, S. Beecham, Tools used in global software engineering: a systematic mapping review, Information and Software Technology (2012).
[13] B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Version, 2(EBSE 2007-001):2007-001, 2007.
[14] O.E. Ariss, D. Xu, S. Dandey, B. Vender, P. McClean, B. Slator. A systematic capture and replay strategy for testing complex GUI based java applications, in: Conference on Information Technology, 2010, pp. 1038–1043.
[15] A.C.R. Paiva, N. Tillmann, J.C.P. Faria, R.F.A.M. Vidal, Modeling and testing hierarchical GUIs, in: Workshop on Abstract State Machines, 2005.

[16] A.M. Memon, M.E. Pollack, M.L. Soffa, Hierarchical GUI test case generation using automated planning, IEEE Transactions on Software Engineering 27 (2) (2001) 144–155.
[17] L. White, H. Almezen, Generating test cases for GUI responsibilities using complete interaction sequences, in: Symposium on Software Reliability Engineering, 2000, p. 110.
[18] L. Baresi, M. Young, Test Oracles. Technical Report CIS-TR-01-02, University of Oregon, Dept. of Computer and Information Science, Eugene, Oregon, USA, August 2001.
[19] J. Takahashi, An automated oracle for verifying GUI objects, ACM SIGSOFT Software Engineering Notes 26 (4) (2001) 83–88.
[20] Q. Xie, A.M. Memon, Designing and comparing automated test oracles for GUI-based software applications, ACM Transactions on Software Engineering and Methodology 16 (1) (2007) 1–36.
[21] A.M. Memon, M.L. Soffa, M.E. Pollack, Coverage criteria for GUI testing, in: Software Engineering Conference held Jointly with ACM SIGSOFT Symposium on Foundations of Software Engineering, 2001, pp. 256–267.
[22] X. Yuan, M.B. Cohen, A.M. Memon, GUI interaction testing: incorporating event context, in: IEEE Transactions on Software Engineering, 2010.
[23] L. Zhao, K.-Y. Cai, Event handler-based coverage for GUI testing, in: Conference on Quality Software, 2010, pp. 326–331.
[24] Z. Hui, R. Chen, S. Huang, B. Hu, Gui regression testing based on function-diagram, in: Intelligent Computing and Intelligent Systems (ICIS), 2010 IEEE International Conference on, vol. 2, October 2010, pp. 559–563.
[25] A.M. Memon, Automatically repairing event sequence-based GUI test suites for regression testing, ACM Transactions on Software Engineering and Methodology 18 (2) (2008) 1–36.
[26] L.J. White, Regression testing of GUI event interactions, in: Conference on Software Maintenance, 1996, pp. 350–358.
[27] http://www.cs.umd.edu/atif/testbeds/testbeds2009.htm.
[28] http://www.cs.umd.edu/atif/testbeds/testbeds2010.htm.
[29] http://www.cs.umd.edu/atif/testbeds/testbeds2011.htm.
[30] http://www.cs.umd.edu/atif/testbeds/testbeds2013.htm.
[31] F.M. Theodore, D. Hellmann, Ali. Hosseini-Khayat, Agile Interaction Design and Test-Driven Development of User Interfaces – A Literature Review, vol. 9, Springer, 2010.
[32] A.M. Memon, B.N. Nguyen, Advances in automated model-based system testing of software applications with a GUI front-end, in: M.V. Zelkowitz (Ed.), Advances in Computers, vol. 80, Academic Press, 2010. pp. nnn–nnn.
[33] http://www.softqual.ucalgary.ca/projects/2012/GUI_SM/.
[34] M.B. Dwyer, V. Carr, L. Hines, Model checking graphical user interfaces using abstractions, in: ESEC/SIGSOFT FSE, 1997, pp. 244–261.
[35] X. Yuan, A.M. Memon, Generating event sequence-based test cases using GUI runtime state feedback, IEEE Transactions on Software Engineering 36 (2010) 81–95.
[36] M.J. Harrold, Testing: a roadmap, in: Proceedings of the Conference on The Future of Software Engineering, ICSE '00, ACM, New York, NY, USA, 2000, pp. 61–72.
[37] S. McConnell, Daily build and smoke test, IEEE Software 13 (4) (1996) 143–144.
[38] X. Yuan, A.M. Memon, Using GUI run-time state as feedback to generate test cases, in: Proceedings of the 29th International Conference on Software Engineering, ICSE '07, IEEE Computer Society, Washington, DC, USA, 2007, pp. 396–405.
[39] V. Basili, G. Caldiera, H. Rombach, Encyclopedia of Software Engineering, John Wiley & Sons Inc., 1994. Chapter Goal Question Metric Approach, pp. 528–532.
[40] H. Zhang, M.A. Babar, P. Tell, Identifying relevant studies in software engineering, Information and Software Technology 53 (2011) 625–637.
[41] N. Abdallah, S. Ramakrishnan, Automated stress testing of windows mobile GUI applications, in: International Symposium on Software Reliability Engineering, 2009.
[42] H. Okada, T. Asahi, GUITESTER: a log-based usability testing tool for graphical user interfaces, IEICE Transactions on Information and Systems 82 (1999) 1030–1041.
[43] M. Safoutin, C. Atman, R. Adams, T. Rutar, J. Kramlich, J. Fridley, A design attribute framework for course planning and learning assessment, IEEE Transactions on Education 43 (2) (2000) 188–199.
[44] F. Belli, Finite-state testing and analysis of graphical user interfaces, in: Symposium on Software Reliability Engineering, 2001, p. 34.
[45] R.K. Shehady, D.P. Siewiorek, A method to automate user interface testing using variable finite state machines, in: Symposium on Fault-Tolerant Computing, 1997, p. 80.
[46] C. Bertolini, A. Mota, Using probabilistic model checking to evaluate GUI testing techniques, in: Conference on Software Engineering and Formal Methods, 2009, pp. 115–124.
[47] K. Magel, I. Alsmadi, GUI structural metrics and testability testing, in: Conference on Software Engineering and Applications, 2007, pp. 91–95.
[48] S. Ganov, C. Killmar, S. Khurshid, D.E. Perry, Event listener analysis and symbolic execution for testing GUI applications, Formal Methods and Software Engineering 5885 (1) (2009) 69–87.
[49] Y. Tsujino, A verification method for some GUI dialogue properties, Systems and Computers in Japan (2000) 38–46.
[50] J. Strecker, A. Memon, Relationships between test suites, faults, and fault detection in GUI testing, in: Conference on Software Testing, Verification, and Validation, 2008, pp. 12–21.
[51] D. Amalfitano, A.R. Fasolino, P. Tramontana, Rich internet application testing using execution trace data, in: Conference on Software Testing, Verification, and Validation Workshops, 2010, pp. 274–283.
[52] A.M. Memon, M.E. Pollack, M.L. Soffa, Plan generation for GUI testing, in: Conference on Artificial Intelligence Planning and Scheduling, 2000, pp. 226–235.
[53] P. Li, T. Huynh, M. Reformat, J. Miller, A practical approach to testing GUI systems, Empirical Software Engineering 12 (4) (2007) 331–357.
[54] B. Robinson, P. Brooks, An initial study of customer-reported GUI defects, in: Conference on Software Testing, Verification, and Validation Workshops, 2009, pp. 267–274.
[55] GUITAR – A GUI Testing frAmewoRk. <http://guitar.sourceforge.net>.
[56] A. Derezinska, T. Malek, Unified automatic testing of a GUI applications' family on an example of RSS aggregators, in: Multiconference on Computer Science and Information Technology, 2006, pp. 549–559.
[57] A. Derezinska, T. Malek, Experiences in testing automation of a family of functional-and GUI-similar programs, Journal of Computer Science and Applications 4 (1) (2007) 13–26.
[58] C. Hu, I. Neamtiu, Automating gui testing for android applications, in: Proceedings of the 6th International Workshop on Automation of Software Test, 2011, pp. 77–83.
[59] C. Hu, I. Neamtiu, A gui bug finding framework for android applications, in: Proceedings of the 2011 ACM Symposium on Applied Computing, SAC '11, ACM, 2011, pp. 1490–1491.
[60] A. Holmes, M. Kellogg, Automating functional tests using selenium, in: agile Conference, 2006, pp. 270–275.
[61] L. Feng, S. Zhuang, Action-driven automation test framework for graphical user interface (GUI) software testing, in: Autotestcon, 2007, pp. 22–27.
[62] Y. Hou, R. Chen, Z. Du, Automated GUI testing for J2ME software based on FSM, in: Conference on Scalable Computing and Communications, 2009, pp. 341–346.
[63] K. Conroy, M. Grechanik, M. Hellige, E. Liongosari, Q. Xie, Automatic test generation from GUI applications for testing web services, in: Conference on Software Maintenance, 2007, pp. 345–354.
[64] V. Chinnapongsea, I. Lee, O. Sokolsky, S. Wang, P.L. Jones, Model-based testing of GUI-driven applications, in: Workshop on Software Technologies for Embedded and Ubiquitous Systems, 2009, pp. 203–214.
[65] M. Alles, D. Crosby, C. Erickson, B. Harleton, M. Marsiglia, G. Pattison, C. Stienstra, Presenter First: Organizing Complex GUI Applications for Test-Driven Development, in: Proceedings of the conference on AGILE 2006, 2006, pp. 276–288.
[66] T. Daboczi, I. Kollar, G. Simon, T. Megyeri, Automatic testing of graphical user interfaces, in: Instrumentation and Measurement Technology Conference, 2003, pp. 441–445.
[67] R. Lo, R. Webby, R. Jeffery, Sizing and estimating the coding and unit testing effort for GUI systems, in: Software Metrics Symposium, 1996, p. 166.
[68] T.-H. Chang, T. Yeh, R.C. Miller, GUI testing using computer vision, in: Conference on Human factors in computing systems, 2010, pp. 1535–1544.
[69] S. Arlt, C. Bertolini, M. Schäf, Behind the scenes: an approach to incorporate context in GUI test case generation, in: Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, ICSTW '11, IEEE Computer Society, 2011, pp. 222–231.
[70] S. Herbold, J. Grabowski, S. Waack, U. Bünting, Improved bug reporting and reproduction through non-intrusive gui usage monitoring and automated replaying, in: Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, ICSTW '11, 2011, pp. 232–241.
[71] X. Yuan, A.M. Memon, Iterative execution-feedback model-directed GUI testing, Information and Software Technology 52 (5) (2010) 559–575.
[72] A.M. Memon, M.E. Pollack, M.L. Soffa, Automated test oracles for GUIs, ACM SIGSOFT Software Engineering Notes 25 (6) (2000) 30–39.
[73] A. Jaaskelainen, A. Kervinen, M. Katara, Creating a test model library for GUI testing of Smartphone applications, in: Conference on Quality Software, 2008, pp. 276–282.
[74] A. Kervinen, M. Maunumaa, T. Paakkonen, M. Katara, Model-based testing through a GUI, Formal Approaches to Software Testing 3997 (1) (2006) 16–31.
[75] M. Navarro, P. Luis, S. Ruiz, D.M. Perez, Gregorio, A proposal for automatic testing of GUIs based on annotated use cases, Advances in Software Engineering 2010 (1) (2010) 1–8.
[76] W.-K. Chen, T.-H. Tsai, H.-H. Chao, Integration of specification-based and CR-based approaches for GUI testing, in: Conference on Advanced Information Networking and Applications, 2005, pp. 967–972.
[77] S. Ganov, C. Kilmar, S. Khurshid, D. Perry, Test generation for graphical user interfaces based on symbolic execution, in: Proceedings of the International Workshop on Automation of Software Test, 2008.
[78] K.-Y. Cai, L. Zhao, H. Hu, C.-H. Jiang, On the test case definition for GUI testing, in: Conference on Quality Software, 2005, pp. 19–28.
[79] Q. Xie, A. Memon, Rapid "Crash Testing" for continuously evolving GUI-based software applications, in: Conference on Software Maintenance, 2005, pp. 473–482.
[80] Q. Xie, A.M. Memon, Using a pilot study to derive a GUI model for automated testing, ACM Transactions on Software Engineering and Methodology 18 (2) (2008) 1–33.

[81] M. Grechanik, Q. Xie, C. Fu, Experimental assessment of manual versus tool-based maintenance of GUI-directed test scripts, Conference on Software Maintenance, 2009, pp. 9–18.
[82] C. McMahon, History of a large test automation project using selenium, in: Proceedings of the 2009 Agile Conference, 2009, pp. 363–368.
[83] R.M. Patton, G.H. Walton, An automated testing perspective of graphical user interfaces, in: The Interservice/Industry Training, Simulation and Education Conference, 2003.
[84] S. Yip, D. Robson, Applying formal specification and functional testing to graphical user interfaces, in: Advanced Computer Technology, Reliable Systems and Applications European Computer Conference, 1991, pp. 557–561.
[85] S. Yip, D. Robson, Graphical user interfaces validation: a problem analysis and a strategy to solution, in: Conference on System Sciences, 1991.
[86] D.H. Nguyen, P. Strooper, J.G. Suess, Model-based testing of multiple GUI variants using the GUI test generator, in: Workshop on Automation of Software Test, 2010, pp. 24–30.
[87] M. Grechanik, Q. Xie, C. Fu, Maintaining and evolving GUI-directed test scripts, in: Conference on Software Engineering, 2009, pp. 408–418.
[88] D. Amalfitano, A.R. Fasolino, P. Tramontana, A GUI crawling-based technique for android mobile application testing, in: Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, ICSTW '11, IEEE Computer Society, 2011, pp. 252–261.
[89] C. Bertolini, A. Mota, E. Aranha, C. Ferraz, GUI testing techniques evaluation by designed experiments, in: Conference on Software Testing, Verification and Validation, 2010, pp. 235–244.
[90] C. Lowell, J. Stell-Smith, Successful automation of GUI driven acceptance testing, Extreme Programming and Agile Processes in Software Engineering 2675 (1) (2003) 1011–1012.
[91] M. Ye, B. Feng, Y. Lin, L. Zhu, Neural networks based test cases selection strategy for GUI testing, in: Congress on Intelligent Control and Automation, 2006, pp. 5773–5776.
[92] B. Daniel, Q. Luo, M. Mirzaaghaei, D. Dig, D. Marinov, M. Pezzè, Automated GUI refactoring and test script repair, in: Proceedings of the First International Workshop on End-to-End Test Script Engineering, ETSE '11, ACM, New York, NY, USA, 2011, pp. 38–41.
[93] Y. Shewchuk, V. Garousi, Experience with maintenance of a functional GUI test suite using IBM rational functional tester, in: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, 2010, pp. 489–494.
[94] C. Bertolini, G. Peres, M. Amorim, A. Mota, An empirical evaluation of automated black-box testing techniques for crashing GUIs, in: Software Testing Verification and Validation, 2009, pp. 21–30.
[95] C. Bertolini, A. Mota, A framework for GUI testing based on use case design, in: Conference on Software Testing, Verification, and Validation Workshops, 2010, pp. 252–259.
[96] A. Jaaskelainen, M. Katara, A. Kervinen, M. Maunumaa, T. Paakkonen, T. Takala, H. Virtanen, Automatic GUI test generation for smartphone applications – an evaluation, in: Conference on Software Engineering, 2009, pp. 112–122.
[97] O.-H. Kwon, S.-M. Hwang, Mobile GUI testing tool based on image flow, in: Conference on Computer and Information Science, 2008, pp. 508–512.
[98] J. Chen, S. Subramaniam, Specification-based testing for GUI-based applications, Software Quality Journal 10 (2) (2002) 205–224.
[99] A.C. Paiva, J.C. Faria, N. Tillmann, R.A. Vidal, A model-to-implementation mapping tool for automated model-based GUI testing, Formal Methods and Software Engineering 3785 (1) (2005) 450–464.
[100] R. Gove, J. Faytong, Identifying infeasible GUI test cases using support vector machines and induced grammars, in: Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, 2011, pp. 202–211.
[101] A.C.R. Paiva, J.C.P. Faria, P.M.C. Mendes, Reverse engineered formal models for GUI testing, Formal Methods for Industrial Critical Systems 4916 (1) (2008) 218–233.
[102] M. Cunha, A. Paiva, H. Ferreira, R. Abreu, PETTool: a pattern-based GUI testing tool, in: International Conference on Software Technology and Engineering, 2010, pp. 202–206.
[103] S. Ali, L.C. Briand, H. Hemmati, R.K. Panesar-Walawege, A systematic review of the application and empirical investigation of search-based test case generation, IEEE Transactions on Software Engineering 36 (2010) 742–762.
[104] N. Juristo, A.M. Moreno, S. Vegas, Reviewing 25 years of testing technique experiments, Empirical Software Engineering 9 (2004) 7–44.
[105] P. McMinn, Search-based software test data generation: a survey: research articles, Software Testing, Verification and Reliability 14 (2004) 105–156.
[106] M. Grindal, J. Offutt, S.F. Andler, Combination testing strategies: a survey, Software Testing, Verification, and Reliability 15 (2005) 167–199.
[107] C.S. Păsăreanu, W. Visser, A survey of new trends in symbolic execution for software testing and analysis, International Journal on Software Tools for Technology Transfer 11 (4) (2009) 339–353.
[108] S.A.M. Mark Harman, Y. Zhang, Search based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications, Technical Report TR-09-03, Department of Computer Science, King's College London, April 2009.
[109] http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/.