A Survey on Developer Tools for the Design, Implementation, Testing, and Maintenance of User Interfaces

Sabiha Salma
Department of Computer Science
George Mason University
Farifax, VA, USA
ssalma@gmu.edu

ABSTRACT

With the increasing use of software applications in today's digital world, user interfaces (UI) have become a crucial component of software to enhance human computer interaction (HCI). The quality of such interactions largely depends on the design and development of software interface tools, which provides the developers with an easy and efficient workflow. However, due to the rapid changes in the functionalities of software artifacts, it is imperative to emphasize the HCI design principles while developing the UI tools. Software engineering (SE) researchers have come up with divergent strategies and tools to ensure best practices for designing an interface. With the purpose of finding possible future directions from this body of research, this report presents a survey of research studies in the intersection of SE and HCI. This survey covers 16 conference and three journal papers across seven distinct research area. This report presents a detailed critical review of existing research studies focused on improving human interaction via developer tools. The outcomes of the study include the software development life cycle (SDLC) taxonomy and research area taxonomy for the classification of the research studies, and a discussion of four problems and possible solutions that can guide future research.

1 INTRODUCTION

Software is ubiquitous around the world and has permeated our lives, whether it's social media networks, applications on our phones, the computers that we use in our everyday life or even in self-driving cars. Software continues to play an ever-increasing role throughout every sphere of our interactive technological activities. Because of this widespread use, software engineering (SE) has become a popular field of research. SE researchers have been conducting numerous studies to enhance the basic functionalities of software that improve users' experiences. Few of their goals include developing programming languages & the associated environments, applying automation in coding & analysis, and ensuring reliability & security while providing simplicity, efficacy & ease of use. These research goals aim at addressing various issues that software developers face while developing software artifacts. One of the major concerns of the developers is to come up with a good design of the software, which is another significant domain for SE research. A good design is incredibly important during the design and development of the software, which is essentially focused on the concept of usability. It is desirable that a user interface should be easy to use and provide efficiency while creating an effective end product to obtain user satisfaction. A constructive approach for improving the usability of any software interface is to practically study methods that deal with how different components of the interface interact with users [24]. For all available user-facing software, it is found that almost half of the code is devoted to the user interface in some manner and as much as 50% of the development effort and 37% of the maintenance effort goes in to making sure that the user interfaces are implemented to interact properly [23]. This report presents a survey that points towards the immense significance of human computer interaction (HCI) design considerations while developing any software interface. HCI is a discipline that focuses on how various interactive computational devices can be made more usable for humans through developing their design, evaluation and implementation [13].

Over years, there has been a lot of work in SE research to support users of different levels with a smooth interaction with devices. The user interfaces (UIs) that we have today were not the same couple of decades ago. At present, we can avail the graphical user interface (GUI) of any device to interact with the underlying operating system through different elements of the interface such as buttons, icons, images, scroll-bars and so on. The current state of GUI dates back to the 1970s when the then research scientists designed the first GUI for Xerox Start workstation [15] which was designed to bring automation in office works. Since then, the interfaces of devices and their underlying software have been greatly evolving through the inventions of desktop computers, laptops, tablets, smartphones and all that enabled the users with more affordable and accessible interactions. With the advent of technological developments, researchers have been focusing on every minute feature and functionality of software that could make it more interactive with the use of efficient and automated approaches.

In a survey [14] of usability automation evaluation in existing technologies, it was found that automation is used only in 33% of surveyed methods that relate to interface use. Based on the utilities of artificial intelligence (AI), the authors of [17] developed an automated technique to determine the behavior of GUI elements. With the advancement of research methodologies, researchers started using various new approaches to address the same problems but with more efficiency. The authors of [22] utilized computer vision (CV) and natural language processing techniques to bring automation for detecting GUI changes in consecutive releases of mobile applications. Deep learning techniques were used by the authors of [4] address automation while analyzing the UI elements. Machine learning techniques were adopted by the authors of [18] for the automation in prototyping of GUI design elements. The authors of [2] came up with a combination of deep learning (DL) and CV techniques and applied them on video data to address bug reporting issues in UIs by generating test scenarios. To aid in automating GUI testing, the authors of [11] utilized genetic algorithms. Throughout

the evolution of research methodologies, it has been observed that the quality of the required dataset has also enriched since all these diverse techniques largely depend on interaction data. However, to ensure a meaningful interaction that fulfills user requirements, the 'human-machine' interface for GUI development tools should be considered with proper significance. Thus, it will be a major focus of this survey to examine this aspect of the studied developer tools.

For the purpose of analyzing and measuring the reliability, authenticity and novelty of any approach proposed by the authors of different articles, it is important to review and scrutinize the content created by the them. After a successful completion of the review process, these methods and techniques get the recognition of a sound approach and appear in various publication venues based on the category and type of the articles. These articles may appear at conferences or as journal articles. Few important conferences for SE research publications include International Conference on Software Engineering (ICSE), Foundations of Software Engineering (FSE), Automated Software Engineering (ASE), Mining Software Repositories (MSR), IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE International Conference on Software Testing, Verification and Validation (ICST) and International Symposium on Software Testing and Analysis (ISSTA). Transactions on Software Engineering (TSE), Transactions on Software Engineering and Methodology (TOSEM), Empirical Software Engineering (EMSE) are few of the renowned venues for journal publications for SE research. ACM Symposium on User Interface Software and Technology (UIST) and ACM Conference on Human Factors in Computing Systems (CHI) serve as two important conference venues for the publication of articles in HCI research domain.

Despite so many extensive research published in various journal and conference venues from both SE and HCI domain, there has been comparatively less work that focuses on how developers actually interact with different interface tools and how to make these automated tools usable. A lot of existing tools help developers in analyzing and formatting the code but, however, there remains a gap between what is displayed on the code and what is displayed on the GUI. Thus, it is challenging for GUI developers and designers to come up with a developer interface that would allow programmers to go from design phase to implementation phase with easy interactions. For an instance, the research works performed to address the issues with design violations [19] or UI to code conversion [18] reveal that there are rooms for improvement in every phase of SDLC when it comes to designing and developing UIs. Therefore, the primary focus of this survey was on the automated tools for the design, development and testing of the UIs. The survey presented in this report is based on 16 conference papers and three journal articles that belong to SE and HCI domain. Through the survey, seven distinct research areas have been covered by generating critical reviews on the articles. The main contributions of this survey include:

(1) SDLC taxonomy of the articles.
(2) Research area taxonomy of the articles.
(3) Strengths, weaknesses and limitations of the articles.
(4) Open research problems for possible future exploration.

The remainder of the report is organized as follows. In section 2, necessary concepts and definitions have been explained to make the rest of the report easily readable. In section 3, the SDLC taxonomy and research area taxonomy have been presented to show how different articles were classified into different SDLC phases and research areas. In section 4, the built taxonomy from section 3 was used to organize the articles with appropriate rationale behind the classification along with a brief summary, strengths and weaknesses of each article. Four open problems that originate from the studied articles and may guide possible future directions have been discussed in section 5. With acknowledgments in section 6, the concluding remarks iare provided in section 7.

2 PRELIMINARIES

2.1 An Overview of Software UIs and Development Tools

UI is the pictorial depiction of any user facing software application. It has, therefore, a very significant role in the development of any such application. A good UI ensures that the users will be able to ascertain what can be accomplished through the interface and what events are taking place as a result of the interaction [25]. Due to the changes in the architecture of software systems, the typical representation of UIs has evolved to a great extent over time. User experiences with different UIs have also changed and vary based on the usability they provide. For example, complexity, cognitive load on users or control over the interface are not the same for a GUI and a touch-screen mobile interface. Researchers have shown their interest to work on different aspects of such interfaces that span over not only SDLC phases but also numerous types of platforms, concepts, techniques, resources and target users.

UIs work as an intermediary medium that enables the users to interact with the software. For technical users such as developers, researchers have focused on different facets of this interaction to provide support. While developing any software, developers use different programming languages in a programming environment that aids them with extensive resources during the development process. There are a number of integrated development environments (IDEs), which were built for the purpose of providing different tools to the developers. While writing codes, developers may interact with the IDE through one or more of these tools. With the purpose of improving this interaction, researchers have worked on how developers can easily get the history of the code changes [12], how they can get a readable code that functions behind data transformation [9] and so on.

Consumption of time is one of the concerns in the software development process. To address this challenging issue, researchers have come up with automated approaches so that developers may rid of the manual efforts. These strategies have covered several research domains leveraging mixed techniques and resources for designing UIs of different platforms. Developing UIs is considered to be a fragment of front end development. Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript are few of the common client side technologies that are used in UI development. Document Object Model (DOM) is how an HTML document is presented in tree structure with its nodes and objects. This structure,

along with JavaScript technology, are used by researchers to interact with the DOM for accessing the corresponding page metadata to support developers with automatically generated code patterns [6]. Automation strategy has been adopted by many other researchers while mining visual, textual, structural and interaction data with various purposes including image-to-code conversion [18], auto-completing bug reproduction steps [20], automated UI analysis [2] and so on.

Exploring the requirements of users, the challenges & problems in existing applications, designing prototypes based on the specifications of the user needs, building the UIs and maintain the interfaces are some of the design processes for the UIs, which fit into different SDLC phases. In many researches, crowdsourcing model was applied where a group of people were involved for achieving the research goal. With the advent mobile devices, SE researchers have broaden their scope of research from web based applications to mobile applications for addressing these design goals. From built mockups to implemented code to GUI based testing to maintenance, all these stages deal with data obtained from the human interaction with UI components. UIs are generally constructed with different components such as buttons, scrollbars, menu, font color and so on. To aid researchers with user interaction data based on these components, repositories have been made to store design data and application based metadata [8]. The types of interaction data and analysis also vary in different research studies. Researchers have done static analysis, dynamic analysis and in some cases both to work on images, videos and GUI animations.

2.2 Computer Vision and Deep Learning for UI Development Tools

As new technologies are emerged, researchers have developed their strategies to address different problems more efficiently. Since interaction data mostly include graphical components, therefore, CV techniques are getting used extensively. On top of AI based approaches, researchers are utilizing machine learning (ML) techniques so that their tools can learn from the existing interaction information to make prediction or informed decisions for unseen data. For larger dataset, researchers are deploying DL techniques as well when it comes to high level feature extraction for tasks like classifying images or processing natural language that belong to the UI. ResNet50 is an example of a DL model, which is pre-trained and performs as an image classifier. In some cases, researchers blend multiple techniques together [18]. In recent times, researchers are also using more powerful techniques like variational auto encoder (VAE) for encoding & decoding data, reinforcement learning (RL) for training agents, neural networks (NNs) for interpretation of data by grouping them, and generative adversarial network (GAN) for generating new synthetic sample of data by utilizing two NNs.

3 AREA TAXONOMY

This report confers the analytical assessment of research papers where authors presented diverse approaches and/or studies to address different problems in existing methodologies regarding how humans interact with developer tools. The authors introduced their approaches and presented the results of their studies before the developers to support them in the design and development process of an effective software interface. This report presents an analysis of the core concept of the papers that relates to one or more SDLC phases. This yielded a taxonomy of all these research studies, which helped in categorizing them through a systematic mapping between the development phase and relevant research area. The articles are therefore classified in both SDLC taxonomy and research area taxonomy. A few of these articles belong to more than one research area and different research areas are repeated across different SDLC phases. To understand how each article relates to a certain category under a taxonomy, it will be helpful to have a general idea about these categories in both taxonomy areas.

3.1 SDLC Taxonomy

For the design and development of software applications, four different abstract representations have been provided in [26] and waterfall model is one of them. This model was chosen to build the SDLC taxonomy presented in this survey because it characterizes the underlying software developing activities as distinct phases [26]. Waterfall model describes following five phases for developing a software.

3.1.1 ***Requirements analysis & definition:*** This is the starting phase in SDLC where the developer team carefully analyzes the requirements of the software users to define their expectations from the application. Through this analysis, the team collects all necessary information that creates the base for the rest of the development cycle. The analysis done in this phase is very crucial to discover and address any probable conflicts in user requirements for the purpose of developing a safe and secure system [26].

3.1.2 ***System & software design:*** The second phase is all about the design of the underlying system architecture. It includes the design of the software application and the design of the UI as well. In this phase, the developers choose to design how different components in the system will interact and how the information will be stored. These design decisions are based on the specified user requirements analyzed in the first phase. In the system design process, developers may choose to come up with prototypes of the application for developing system UI [26].

3.1.3 ***Implementation & unit testing:*** The third phase of the model is mainly handled by the programmers who write codes to put different units of the design into effect and verify them. Based on the architecture, the functionalities of a software application are performed by multiple sub-systems that are responsible for implementing different components of the whole application. Unit testing is done for the validation of these components [26]. Therefore, developers focus on the implementation and testing of these sub-systems in this phase.

3.1.4 ***Integration & system testing:*** Separate components of the software, which are implemented and tested individually in the third phase, are assembled and go through a full-length testing in the fourth phase of the model [26]. During this phase, different functions of various components of the software are combined into one application.

3.1.5 ***Operation & maintenance:*** This is the final phase of the developing process. In this phase, the final product is shared with

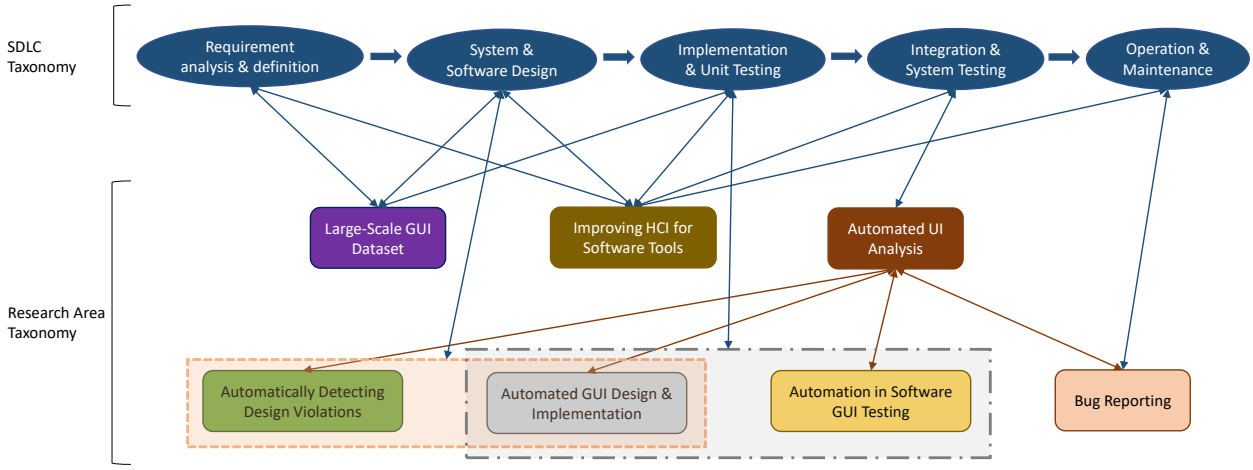

Figure 1: Illustration of the Derived Taxonomy

end users so they can have hands-on experience with the application and help developers in identifying the flaws and bugs in the system [26]. Through repeated evaluation and evolution of the software, the improvement of the application is maintained.

3.2 Research Area Taxonomy

Based on the articles, seven distinct research areas were found while building the taxonomy. Multiple articles have fallen into same category based on the core concept carried out by the paperwork. Figure 1 presents a compact hierarchical representation of the taxonomy areas. Based on the built taxonomy, a detailed description of the survey is presented in section 4 after a brief introduction of different areas of research.

3.2.1 ***Large-scale GUI dataset:*** GUI datasets are of great significance in SE research that are used to address different problems in user-facing software applications [8]. For the purpose of developing interacting applications, researchers extensively use these datasets that interpret the attributes and characteristics of designs of existing software.

3.2.2 ***Improving HCI for software tools:*** Users' interactions and experiences with software applications are closely connected to each other. It is a fundamental requirement for a usable and successfully functional software application to provide developers with interactive tools. This particular research area has been becoming an emerging field for researchers who focus on problems that developers face in all phases of SDLC.

3.2.3 ***Automated UI analysis:*** Automated analysis of UI components has been considered as a fundamental research perspective. Studies based on automated UI analysis further spans across research directions that include automation in design violation detection, GUI design & implementation, GUI testing and so on.

3.2.4 ***Automated bug reporting:*** Due to increasing use of software applications, documentation on bug tracking has been a significant area for SE research [21]. Researchers have been exploring diverse aspects while analyzing UI components to reduce the complication in application maintenance.

3.2.5 ***Automatically detecting design violations:*** It is a common problem in software applications that the final design of the software product does not reflect the original design [19]. This particular area of research addresses the issue of inconsistency between the planned design and implemented design of applications and directs towards automatic violation detection approaches to resolve the issue.

3.2.6 ***Automating GUI design and implementation:*** The interpretation of GUI designs plays an important role in conveying the developers with necessary information that they can use to recreate and/or reuse different UI components. Researchers have been working in this area to support developers with automatic approaches for generating codes or designing UIs.

3.2.7 ***Automation in software GUI testing:*** There has been quite a number of research studies focused in automating the testing process when it comes to GUI testing. Researchers have come up with various tools and methodologies to address different testing issues.

4 TAXONOMY-BASED SURVEY

This section presents the rationale behind the classification of each article into SDLC taxonomy and research area taxonomy. This section also presents a brief summary of each article and its strength & weakness. As mentioned in section 3, one particular research area might be recurrent in different SDLC phases based on the research methodologies, tools, techniques and approaches presented by the authors in different articles.

4.1 Requirements Analysis & Definitions

Articles of two categories from the research area taxonomy fall under the first SDLC phase. These categories are further presented with associated articles and their findings.

4.1.1 ***Large-scale GUI dataset:*** The authors of [8] built a repository of mobile application design data by gathering unique UIs together and named the dataset as RICO. While building RICO, the authors focused on Android applications and collected the design data & metadata from Google Play Store. While mining visual, textual, structural and interaction data, the authors combined powerful methodologies like crowdsourcing and automation that landed them with a strong research. The collection of 72,219 unique UIs spanning over 27 different categories of Android applications serves as one of the strengths of the dataset. Moreover, the authors trained an autoencoder to learn the representation of each UI in their repository for enabling DL applications. However, a limitation of this research work is the use of only Android applications for building the repository. Despite the richness of the dataset, a considerable weakness of this research work is that the authors chose the runtime of the automated explorer as the limiting factor while measuring and comparing performances of their hybrid system with that of human and automated exploration individually. Another set of experiment could possibly be done by setting the coverage percentage as the limiting factor instead of runtime to see if the results of tho experiments come out different.

4.1.2 ***Improving HCI for software tools:*** Improvement in the design and development of interaction tools in user-facing software are of great concern, may it be for providing transformation and mapping of useful data based on archived data [9] or for multimedia authoring, sketching and so on that are used by practitioners [3].

The authors of [9] presented a user-driven interaction model named Wrex in their research that performs as a Jupyter extension and generates readable code in Python. This article is about transforming data from one form to another. For the implementation of Wrex, the authors utilized an existing interactive grid view and modified it to render views of the underlying dataframe. The program synthesis engine in Wrex extends an existing toolkit that supports transformation of string, number, date and table lookup. The authors conducted a comparative usability study with 12 data scientists and collected qualitative feedback from the participants. The strength of the approach discussed in this article lies in addressing the need of data scientists who require inspection of the code that transform their data. Moreover, the authors applied program synthesis for a larger dataset in a scalable manner, which was confined to limited scope. However, a shortcoming of this approach that was observed while reading this article was that the authors did not clearly mention how their system would work to handle any missing, inconsistent or noisy data. Moreover, a limitation of this tool is that the authors designed the ranker in such a way that it would prefer only small programs. A further exploration with a DL model could be considered for larger programs. Also, an ML prediction algorithm could be used to train the tool on noisy data so that it could handle such dataset.

Another article, which relates to the first phase of SDLC and belongs to the research area of improving HCI for software tools, is [3]. This journal article is about supporting practitioners with design tools while transitioning workstyles. The authors defined *workstyle* as the style of the developers while they interact with the system. For example, some developers may have preferences of tools or environment setup while they edit their code. Therefore, when they need to transform their working environment, it is important for them to ensure a convenient work style. For this, the authors conducted a survey with 370 stakeholders in software development paradigm. In this survey, they provided several example scenarios of transformation in workstyles. The authors then designed a questionnaire and presented a demographics and survey results based on the answers received from respondents. Based on the survey results, the authors built two design tools named TaskSketch and CanonSketch. TaskSketch aims at supporting practitioners with traceability in workstyle. On the other hand, CannonSketch provides smooth transition to the practitioners. The strength of this article is the questionnaire which forms the basis for designing these two tools. A minor shortcoming of this article is the vague presentation of the usage of tools in different development processes. A more comprehensible demonstration would have been more helpful the readers to easily understand the difference of tool usage among the development processes.

4.2 System & Software Design

Following four research areas are associated with different articles that address system and software design issues.

4.2.1 ***Large-scale GUI dataset:*** As already mentioned in section 4.1.1, the authors of [8] came up with RICO, a repository of mobile application interaction data. The fact that this article deals with design properties of unique UI screens makes it closely relevant to the system and software design phase.

4.2.2 ***Improving HCI for software tools:*** The authors of [6] focused on an autocomplete feature to maintain consistency in HTML code patterns. They proposed their approach through a prototype tool named IRIS that facilitates developers with autocomplete suggestions by enabling them to interact with code structure and creating code patterns while editing HTML documents. The whole approach was presented in three sections- (a) learning code patterns, (b) using code patterns to suggest completions and (c) interaction with code patterns. For evaluation, the authors recruited 24 participants who were asked to complete any two of the three assigned tasks- creation, continuation and correction. Their approach is strong due to the use of ML technique for the purpose of feature extraction. However, the authors did not mention about how long does it take for the system to generate a rule or decision while creating the code patterns. Since they used a JavaScript implementation of ID3 algorithm, one of its weaknesses is that only one attribute at a time can be tested for making a decision. To address this probable, another algorith named as C4.5 could be used instead of ID3, which is a preferred method in ML [10]. Ross Quinlan developed this algorithm as an extension of the ID3 algorithm, also developed by him. Both algorithms are utilized for building decision trees for the purpose of classification of data. C4.5 algorithm provides better accuracy and less time consumption during execution as mentioned in [10].

Reference [3] on workstyle transitioning, briefly summarized in section 4.1.2, is correlated with system and software design phase as well. The survey conducted by the authors was focused on the transformation of workstyles in regards to UI fashions. This article

is placed under the above mentioned research area since it is all about software tools that practitioners interact with.

4.2.3 ***Automatically detecting design violations:*** The authors of [19] presented an automated approach to measure the gap between the application UI and original design. For this, they designed a tool named GVT to provide automation in reporting design violations. The working strategy of GVT is comprised of three main stages- i) GUI collection phase, ii) GUI comprehension phase and iii) design violation detection phase. The authors conducted an empirical study based on three research questions for the purpose of measuring the performance, usefulness & industrial applicability of their proposed approach. To measure performance, precision & recall were used as the evaluation metrics. One of the strengths of this article lies in the use of K-Nearest-Neighbors (KNN), an efficient ML based algorithm in the second phase and CV techniques in the third phase. The fact that GVT is currently being used by over 1000 Huawei professionals also represents the strength of the approach. Moreover, the study results with 99.4% for overall detection precision, 98.4% for average classification precision and 96.5% for average recall are evidences of the strength of GVT. However, a minor shortcoming of the article is the lack of detailed explanation of the drawbacks mentioned in result section regarding industrial applicability. Also, experiments were based on only static effects which reflects a trivial weakness. This could be addressed by extending the work by including dynamic components.

Another research under the same taxonomy area is referred by the authors as Seenomlay [28] where they addressed the linting problem of GUI animations against UI design guidelines. They proposed a CV & DL based unsupervised method that uses unlabeled real-app GUI animations for learning to generate them. The authors considered the problem as a multi-class classification problem that they solved by applying an ML based algorithm, KNN and VAE network. They used 3 types of inputs- (a) linted GUI animation screencast to validate against the design rules, (b) labeled data for each rule in design-don't guidelines and (c) unlabeled GUI animation screencasts collected from real applications. The authors conducted experiments on this based on four research questions. They made use of CV, ML and DL based techniques that gives strength to the research. Besides, the use of real and synthetic labeled data and unlabeled real-app data for the implementation serves as another strong side of the approach. That said, a weakness of this research is that very few design don't guidelines were used. Also, there were no explicit and detailed explanation on why those specific guidelines were chosen. An adequate and precise clarification would have been useful regarding this.

4.2.4 ***Automated GUI design & implementation:*** The authors of [18] came up with an automated approach in a system called ReDraw for the design and implementation of GUI components in Android applications. They presented an automated GUI prototyping process to reduce the image-to-code abstraction gap. They addressed the challenges of translating GUI components into code that are seen in existing techniques. These challenges include time consumption, errors, collaboration in teams, costs, design violations and so on. To address these issues, ReDraw automatically performs this transformation process, which is comprised of three phases- detection, classification and assembly of GUIs. For the GUI detection phase, the authors used mockup metadata along with CV techniques. For GUI classification, they utilized dynamic analysis in an automated way and leveraged the deep convolutional neural network. For automatic assembly of the GUI components, the authors used KNN algorithm. On top of these powerful technologies used in building ReDraw, the authors conducted an empirical study with four research questions for measuring the accuracy, efficiency, similarity and consideration of ReDraw with existing system. The authors utilized very efficient and powerful technologies to build ReDraw that makes this article very strong. A shortcoming of this approach is that the authors considered only Android applications.

Rewire is another automated interactive technique that was presented by the authors of [27]. This technique enables the UI designers to leverage existing examples of interface designs to reuse them or redraw their UI components. With this purpose, the authors facilitated the designers with three design assistance modes after processing the input screenshots through segmentation, vectorization and beautification in a pipelined mechanism. The authors focused only on primitive shapes that include rectangles, circles, lines & text and used them in a tool for enabling designers to edit, resize or move them. For segmentation, they used CV algorithms. They measured the overall performance of their approach with precision, recall and F-1 score for two evaluation metrics named as type detection and property accuracy. The authors also designed an empirical study to address the accuracy and efficacy of the designers and their feedback in regards to the assistance mode. The combination of the designed tool, ReWire, and the empirical study along with the use of CV techniques serve as the strength of this article. A shortcoming of this work is that there is now way back to modification once the designer is done with constructing the design. To address this limitation for the purpose of further extending the work, a brief explanation of a possible scope of research is given in section 5.3

4.2.5 ***Automated UI analysis:*** Reference [4] addresses the object detection problem for GUI elements. The authors conducted a large-scale empirical study to detect them. For the study, they used seven baseline methods and designed three research questions with precision, recall & F1-score as evaluation metrics. Based on the findings from the study, the authors used over 50K screenshots from RICO dataset [8] and came up with a GUI-based strategy to detect both non-text GUI elements and GUI texts. They adopted a DL based technique to detect GUI texts along with the ResNet50 classifier. For non-text GUI elements, they adopted a 2-stage pipelined architecture comprised of region detection and region classification. The strength of the approach presented in this article is easily comprehensible given the use of large amount of data. In addition to this, the workflow is also what gives strength to this research because the designed approach was properly planned on the basis of solid justification that was achieved through the study. One shortcoming of this article is that an adequate explanation was not provided by the authors regarding the selection of 15 GUI elements that were considered for the study. It would have been useful to know the selection criteria.

4.3 Implementation & Unit Testing

Following three research areas are associated with different articles that address issues related to implementation & unit testing.

4.3.1 ***Improving HCI for software tools:*** As already mentioned in section 4.1.2, an interaction model named Wrex was proposed in [9], which is based on two design goals. The first design goal requires that the implementation has to ensure the availability of the tool within program environment. The second design goal requires that the produced code blocks have to be inspectable. Both these goals are associated with the implementation and unit testing phase of SDLC.

Reference [6], briefly summarized in section 4.2.2, is based on an approach in a tool named IRIS that brings human efforts and AI techniques together for generating code patterns in HTML documents. To accomplish this research goal, different components from the same document are used to build a design tree. Therefore, this research work clearly justifies how it is tagged with implementation phase.

Reference [1] is the third article within the current SDLC and research area taxonomy that deals with the use of annotations to reduce the cost of code comprehension. The authors of this article pointed at the limitations of traditional IDEs in regards with the understanding about how different pieces of a software are related to each other. To address this limitation, the authors presented an empirical study using a non-traditional IDE called Synectic built by them and a traditional IDE Eclipse. In the study design, the authors examined the benefits of annotations for code comprehension. For this, they came up with 2 research questions. The first question addressed the effects of code comprehension regarding increase in accuracy, reducing time, and reducing cognitive load. The second question examined the usability of the annotations for newcomers. A shortcoming of the article is that the authors had the tasks focused on none but an open source Java project named Vaadin Bakery App. The strength of this approach is that the authors not only built the non-traditional IDE named Synectic but also proved through the experiments that it performed better than the traditional IDE Eclipse.

4.3.2 ***Automated GUI design & implementation:*** In the summaries of the techniques ReDraw [18] and ReWire [27] presented in section 4.2.4, it has been discussed that the former one deals with prototyping and the latter one with object extraction. Prototypes are unfinished implementations of the end product and object extraction task in the corresponding article was performed through reverse engineering. Therefore, it is justified that both these articles were classified within the current taxonomy area.

4.3.3 ***Automation in software GUI testing:*** The authors of [11] worked on a data oriented strategy for exploring mutation testing on Android applications. The overall workflow of their approach is divided into three phases. In the first phase, they built a taxonomy of 262 faults originating from 14 groups out of which, 4 categories were relevant to Android. Using this taxonomy in the second phase, they built two mutant generating frameworks named MDroid+ and MutAPK based on 38 derived mutation operators. While addressing this issue, they considered both availability and unavailability of the source code. Therefore, they designed the tools for both source code level and APK level implementation. In the last phase of their approach, the authors conducted an empirical study with four research questions for the purpose of comparing their tools with existing tools. The built taxonomy and the derived mutation operators serve as the core of the entire approach and thus, this particular part of the article serves as the strength. A minor shortcoming of this article is that the underlying research was entirely based on only Android applications.

4.4 Integration & System Testing

Following four research areas are associated with different articles that address issues related to integration & system testing.

4.4.1 ***Automatically detecting design violations:*** In section 4.2.3, the automated approach named GVT was briefly discussed, which was presented by the authors of [19] to report GUI design violations. This article is also tagged with the integration and system testing phase. Because, during the design violation detection phase of the three-fold working strategy, multiple methods were integrated to detect design violations leveraging spatial GUI component information & CV techniques.

4.4.2 ***Automated UI analysis:*** In section 4.2.5, a brief summary of the article [4] has already been presented. The authors of this article conducted an empirical study with seven baseline methods for the purpose of detecting GUI elements considering the images as a whole instead of components. Also, this approach achieved better coverage of the GUI elements. Therefore, it is justified to tag this article within the current taxonomy areas.

Reference [17] presented a test oracle using automation technique to automate the creation of test data by observing the behavior of a GUI. The authors implemented the oracle in their GUI testing system called PATHS. To explain a formal definition of a GUI, they first modeled it with different basic components such as button, scrollbar, text and so on and then they set values to these components. The defined actions for those components and provided a formal description of a test case which is pair of state and action of those components. The authors provided examples to show how GUI models can be utilized to get an expected state of any particular component. The authors described the oracle designer that forms the model of a GUI. Based on AI technique, the automated oracle presents the expected state of different GUI components. The GUI model generates test cases that are run against the GUI and the GUI behavior is tested by the oracle. The authors also designed and developed a system for the execution of systems. They carried out an experiment to measure the time it takes to formulate the state information for any GUI component and to observe the verification process for the tester against the GUI behavior. A weakness of this oracle is that it is required to execute it on the same device as the one within test environment, where it can easily access the GUI of the system. However, the strength of this article lies in the automated approach that the authors presented.

4.4.3 ***Automation in software GUI testing:*** As per the brief description of the approach given in section 4.3.3 presented by the authors of [11], they built a taxonomy for mutant generation to provide an aid in testing the GUIs of Android applications at both source code and APK level integrating all the components. It is,

therefore, best applicable for this article to be categorized in the current taxonomy areas as well.

The authors of [7] presented an automated approach for Android applications that automatically learns ways to interact with different GUI components. The authors used multi-armed bandit (MAB) problem as a test generating template and enhanced RL to train their model for the purpose of generating tests. They carried out the implementation of their approach in form of plug-ins of DroidMate2 that serves as a test input generator for Android. For the evaluation of their approach, the authors came up with three research questions. The main goal of this experiment was to measure the coverage attained by their approach. The use of advanced techniques like RL is definitely a strength of this research. For implementing RL strategies in complicated environments like GUI testing environments, the agents need to ensure their ability to choose only relevant details. However, this article has a shortcoming in regards with a detailed explanation about this requirement. Moreover, a weakness of this approach is the dependency on DroidMate2. In case of any changes to DroidMate2, it might be possible that the whole approach breaks apart. To address this issue, coming up with an independent tool might be a safer way to implement the process.

Reference [16] deals with automated software testing for Android mobile applications. The authors presented a testing tool called Sapienz with a target of maximum code coverage and fault revelation with minimum length of fault revealing test sequences. For the implementation, the authors followed search-based technique and presented three algorithms. The first one represents Sapienz's top-level algorithm which is used to optimize code coverage, sequence length and number of crashes found using Pareto optimal search-based SE approach. The authors defined a test suite operation via the second algorithm that applies crossover, mutation and reproduction operators on each test suite. The third algorithm aims at discovering complex interactions between events that can be invoked from the UI during automated testing. The authors presented six research questions to measure reliable replication, code coverage, fault revelation, sequence length, statistical significance & effect size, and usefulness. Sapienz is implemented on a search-based technique and it performs both static and dynamic analysis for instrumentation which are strengths of this research. A shortcoming of this article is that the authors did not mention anything about how the interface of underlying framework named DEAP is connected with that of Sapienz. Given that Sapienz was implemented entirely on this framework, any issues with the framework will end up with creating issues for Sapienz as well.

Reference [5] presents a comparative study of existing test input generation tools. The authors evaluated the effectiveness of these tools on 60 real-world Android applications based on four metrics- (i) code coverage, (ii) ability to detect faults, (iii) ability to work on multiple platforms, and (iv) ease of use. The study aims to find out the weaknesses and strengths of different approaches that address static and dynamic analysis of test input generation. Based on different strategies that testing tools generate for event-driven inputs, the authors categorized these tools into three categories, which are- (i) random exploration strategy, (ii) model-based exploration strategy and (iii) systematic exploration strategy. In addition to the detailed explanations of each of the tools in these three categories, the authors presented a table that provides an overview of the tools and compares them based on their availability, ability to perform platform/APP instrumentation, ability to generate UI/System events, used strategy, dependency on source code, and used testing strategy. The authors evaluated the tools based on four criteria- (i) effectiveness of the exploration strategy, (ii) fault detection ability, (iii) ease of use, and (iv) android framework compatibility. The meticulous study of the tools along with a concise summary gives strength to this article. However, one weakness of this article is that the authors compared the coverage of the best run out of 10 runs for each tool. There remains a possibility that some of these tools might be giving their best coverage within 10 runs but some might need more than 10 runs. One possible way to address this situation is to consider the percentage of coverage as the limiting factor instead of runs.

4.4.4 ***Automated bug reporting:*** To facilitate developers with automation in creating bug reports for the analysis of visual artifacts, the authors of [2] presented an automated CV-based tool named V2S. The architecture of V2S consists of three phases, which are- (i) touch detection phase, (ii) action classification phase and (iii) scenario generation phase. The authors also performed an empirical study with five research questions for the purpose of assessing the accuracy, robustness, performance and industrial utility of their approach. The evaluation regarding the usefulness of V2S was based on the feedback from only three affiliated developers was not persuasive enough to confirm the generality of the tool. However, the combination of the creation of the automated tool and the empirical study served as a strong mechanism that enabled the tool to accurately replay scenarios from screen recordings. Since this approach could be used for crowdsourcing functional & usability testing and for the creation & maintenance of automated GUI-based test suites, therefore, the placement of this article under this taxonomy is well justified.

4.5 Operation & Maintenance

4.5.1 ***Improving HCI for software tools:*** The authors of [12] addressed inconsistencies in messy computational notebooks by introducing a suite of interactive tools named as code gathering tools. The tools archives all versions of code outputs to recover the subset of code that produced them through a process of finding, cleaning and comparing versions of code using program slicing technique. The authors conducted a qualitative in-lab usability study with 200 randomly selected data analysts where 12 participated in the study. The strength of this work is that the introduced tool enables the program analysts to have the entire history of code changes in form of archived versions. However, program slicing technique offers both static and dynamic analysis. That said, it is a shortcoming of the approach described in this article that it is based on only static analysis. Since data itself is not static, therefore, using both static and dynamic analysis would make this work stronger.

Reference [1] falls under the same taxonomy areas as well. As per the brief summary mentioned in section 4.3.1, this article is focused on code comprehension. This is entirely related to the maintenance of the existing version of the application which justifies the selected taxonomy for this article.

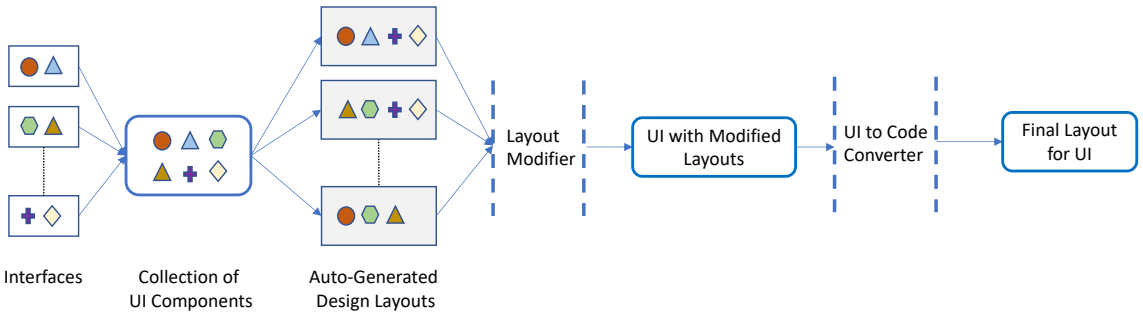

Figure 2: Illustration of the possible solution for enabling two-step modification on automatically generated UI design layouts

4.5.2 ***Automated UI analysis:*** Reference [22] is about detecting and reporting GUI changes during subsequent mobile application versions where the authors proposed an automated approach named GCAT. The authors presented a taxonomy referring to three main categories of changes which affects text, layout and resource. The authors presented GCAT with three main components- (a) version control integration, (b) automated GUI based exploration and (c) GUI interpretation and change detection.In addition to the development of this tool, the authors conducted a study to evaluate GCAT's matching & filtering, performance, effectiveness and developers' usefulness of GCAT reports, which strengthens the research study. Automatic detection of GUI changes, the built taxonomy, and the capability of generating natural language summaries also serve as the strength of this article. Despite these, a limitation of the work is that it was applied on only Android applications. Besides, a weakness of the study was that it was done with only participants from affiliated institution.

4.5.3 ***Automated bug reporting:*** The authors of [21] proposed a bug reporting mechanism called Fusion that helps to retrieve actionable information about a bug by autocompleting the reproduction steps for aiding the creation of explanatory bug reports. Fusion aims at reducing the gap between the bug reporters and developers. The fact that Fusion is completely automatic and its data collection phase is based on both static and dynamic analysis gives it strength. This approach was limited to Android applications only. A shortcoming of the approach is that the authors did not provide any detailed explanation of the type of bugs that were addressed via this approach.

As mentioned earlier in section 4.4.4, the authors of [2] worked on video recordings and generated replayable test scenarios to aid in creation of bug reports, which justifies the categorical placement of this article within the current taxonomy.

5 OPEN PROBLEMS & FUTURE RESEARCH

In section 4, article summaries from different taxonomy areas were briefly discussed along with their strengths, weaknesses and limitations. Based on the discussion points, four out of seven research areas have been approached with a problem statement. This section briefly presents these problems with possible solutions.

5.1 Large-scale GUI dataset

5.1.1 ***Problem statement:*** As mentioned in section 4.1.1, the authors of [8] presented RICO dataset covering a wide range of unique UIs. For the purpose of collecting wider range of interaction data, researchers currently do not have any merged dataset including mobile applications and web applications as two principal categories. Moreover, to open up opportunities to address issues in both Android and iOS platforms while considering mobile applications, a repository with combined dataset is needed.

5.1.2 ***Possible solution:*** A possible solution is to collect the interaction data for each platform, may it be from web-base and mobile applications or iOS and Andoird platforms for mobile applications. Based on the collected interaction data, a system can be designed that would learn from user interactions and predict the next possible interaction. This approach could be used in software security to mitigate manual efforts.

5.2 Improving HCI for software tools

5.2.1 ***Problem statement:*** In section 4.2.2, a brief summary of [6] is already provided where the authors designed an approach in a prototype tool named IRIS. This research work was conducted focusing only on the interaction of developers with HTML documents. It is not clearly mentioned in the article about how the system would handle the cold-start problem, when a developer tries to write a document from scratch. Although this issue cannot be considered as a limitation or weakness of the article, however, it directs towards a possible scope of research with the UI components.

5.2.2 ***Possible solution:*** I intend to get involved in this particular research idea which can be entitled as "Leveraging the UI to aid program comprehension." A preliminary outline of the working strategy might include three steps. In the first step, the UI could be transformed into its basic components. In the second step, ML or even DL techniques could be used to form a tree structure or hierarchy of those components based on how they were interlinked with each other. Learning from existing UIs, different UI patterns could be created in the third step. These patterns could work as autocomplete suggestions for the UI designers while working on an existing UI or creating a new UI. A set of rules to form these suggestions could be generated by mining repositories of interaction data as in [8].

5.3 Automating GUI design and implementation

5.3.1 ***Problem statement:*** In [27], the layouts cannot be modified once they are generated. To address this limitation, I would like to combine the strategies mentioned in [27] and [18] to aid the developers with the ability to modify their layouts and enable them to look into the generated codes if needed.

5.3.2 ***Possible solution:*** The potential research idea might be a two-fold approach. The first unit of the strategy would focus on prediction of possible design layouts. For this, all the outputs from each mode discussed in [27] could be merged together. Then, based on the components of the images, these outputs could be used to predict all possible designs to present before the UI designers. From that point, designers might choose a specific layout of interest and modify according to their expectations. The second unit of the strategy would focus on providing usability during this modification. Designers might want to see the interpreted code blocks for the design layout. For this purpose, the approach in [18] could be utilized to automatically generate the code for the associated design. Figure 2 further illustrates the solution strategy. The UI components from existing examples of interfaces will be collected and combined. Then, an automated approach as mentioned in ReWire [27] might be used to generate design layouts. Next, using the layout modifier, UI designers could modify the generated layouts. Using the image-to-code approach as discussed in ReDraw [18], the designers could further modify the codes to bring changes in the design layouts.

5.4 Automation in software GUI testing

5.4.1 ***Problem statement:*** In existing studies, authors have come up with tools to provide maximum code coverage [16] or conducted comparative studies of existing test input generation tools [5]. Some authors have proposed automated approaches for learning how to interact with different GUI components [7] and some have built taxonomy for mutant generation to provide an aid in testing the GUIs of Android applications [11]. However, the interfaces of many of the existing techniques are fairly poor. The generated logs from them are not very informative. It would be of much more help for the GUI designers and developers if these logs or tracing information could be connected to a debugging framework.

5.4.2 ***Possible solution:*** A feasible approach might be a debugging interface that would allow the developers to step through the programs that take place in their code through visual components. This interface would show how certain pieces of the program change with the modification in the UI.

6 ACKNOWLEDGEMENT

I am grateful for the insightful and timely feedback suggested by the anonymous reviewers of Writing Center, George Mason University.

7 CONCLUSION

Over the past few years, powerful techniques and tools have been implemented by SE researchers to enhance the interaction between humans and software interfaces. This advancement is obligated to improvements in GUI design, implementation and testing. In this survey report, I presented a short description of different aspects and adopted technologies associated with GUI development. I built an SDLC taxonomy and research area taxonomy based on 16 conference and three journal articles and classified them into associated taxonomy area. I critically reviewed the articles and presented a brief summary along with their strengths, shortcomings and weaknesses. Finally, addressing the limitations of few of the reviewed articles, I suggested four open problems indicating future research directions for GUI design, development and testing leveraging GUI dataset.

REFERENCES

[1] Marjan Adeli, Nicholas Nelson, Souti Chattopadhyay, Hayden Coffey, Austin Henley, and Anita Sarma. 2020. Supporting Code Comprehension via Annotations: Right Information at the Right Time and Place. 1–10. https://doi.org/10.1109/VL/HCC50065.2020.9127264

[2] Carlos Bernal-Cárdenas, Nathan Cooper, Kevin Moran, Oscar Chaparro, Andrian Marcus, and Denys Poshyvanyk. 2020. Translating Video Recordings of Mobile App Usages into Replayable Scenarios. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 309–321. https://doi.org/10.1145/3377811.3380328

[3] Pedro Campos and Nuno Jardim Nunes. 2007. Practitioner Tools and Workstyles for User-Interface Design. *IEEE Softw.* 24, 1 (Jan. 2007), 73–80. https://doi.org/10.1109/MS.2007.24

[4] Jieshan Chen, Mulong Xie, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, and Guoqiang Li. 2020. Object Detection for Graphical User Interface: Old Fashioned or Deep Learning or a Combination? *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1202–1214. https://doi.org/10.1145/3368089.3409691

[5] S. R. Choudhary, A. Gorla, and A. Orso. 2015. Automated Test Input Generation for Android: Are We There Yet? (E). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 429–440. https://doi.org/10.1109/ASE.2015.89

[6] Kartik Chugh, Andrea Y. Solis, and Thomas D. LaToza. 2019. Editable AI: Mixed Human-AI Authoring of Code Patterns. *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (Oct 2019). https://doi.org/10.1109/vlhcc.2019.8818871

[7] Christian Degott, Nataniel P. Borges Jr., and Andreas Zeller. 2019. Learning User Interface Element Interactions. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Beijing, China) *(ISSTA 2019)*. Association for Computing Machinery, New York, NY, USA, 296–306. https://doi.org/10.1145/3293882.3330569

[8] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications *(UIST '17)*. Association for Computing Machinery, New York, NY, USA, 845–854. https://doi.org/10.1145/3126594.3126651

[9] Ian Drosos, Titus Barik, Philip J. Guo, Robert DeLine, and Sumit Gulwani. 2020. Wrex: A Unified Programming-by-Example Interaction for Synthesizing Readable Code for Data Scientists. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376442

[10] Halima Elaidi, Zahra Benabbou, and Hassan Abbar. 2018. A Comparative Study of Algorithms Constructing Decision Trees: ID3 and C4.5. In *Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications* (Rabat, Morocco) *(LOPAL '18)*. Association for Computing Machinery, New York, NY, USA, Article 26, 5 pages. https://doi.org/10.1145/3230905.3230916

[11] Camilo Escobar-Velasquez, Mario Linares-Vásquez, G. Bavota, Michele Tufano, K. Moran, M. D. Penta, Christopher Vendome, Carlos Bernal-Cárdenas, and D. Poshyvanyk. 2020. Enabling Mutant Generation for Open- and Closed-Source Android Apps. *IEEE Transactions on Software Engineering* (2020), 1–1.

[12] Andrew Head, Fred Hohman, Titus Barik, Steven M. Drucker, and Robert DeLine. 2019. Managing Messes in Computational Notebooks. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3290605.3300500

[13] Thomas T. Hewett, Ronald Baecker, Stuart Card, Tom Carey, Jean Gasen, Marilyn Mantei, Gary Perlman, Gary Strong, and William Verplank. 1992. *ACM SIGCHI Curricula for Human-Computer Interaction.* Technical Report. New York, NY, USA.

[14] Melody Y. Ivory and Marti A Hearst. 2001. The State of the Art in Automating Usability Evaluation of User Interfaces. *ACM Comput. Surv.* 33, 4 (Dec. 2001), 470–516. https://doi.org/10.1145/503112.503114

[15] Jeff Johnson, Teresa L. Roberts, William Verplank, David C. Smith, Charles H. Irby, Marian Beard, and Kevin Mackey. 1989. The Xerox Star: A Retrospective. *Computer* 22, 9 (Sept. 1989), 11–26, 28–29. https://doi.org/10.1109/2.35211

[16] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-Objective Automated Testing for Android Applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis* (Saarbrücken, Germany) *(ISSTA 2016)*. Association for Computing Machinery, New York, NY, USA, 94–105. https://doi.org/10.1145/2931037.2931054

[17] Atif M. Memon, Martha E. Pollack, and Mary Lou Soffa. 2000. Automated Test Oracles for GUIs. In *Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering: Twenty-First Century Applications* (San Diego, California, USA) *(SIGSOFT '00/FSE-8)*. Association for Computing Machinery, New York, NY, USA, 30–39. https://doi.org/10.1145/355045.355050

[18] Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. 2018. Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps. *IEEE Transactions on Software Engineering* PP (02 2018). https://doi.org/10.1109/TSE.2018.2844788

[19] Kevin Moran, Boyang Li, Carlos Bernal-Cárdenas, Dan Jelf, and Denys Poshyvanyk. 2018. Automated Reporting of GUI Design Violations for Mobile Apps. In *Proceedings of the 40th International Conference on Software Engineering* (Gothenburg, Sweden) *(ICSE '18)*. Association for Computing Machinery, New York, NY, USA, 165–175. https://doi.org/10.1145/3180155.3180246

[20] Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. 2015. Auto-Completing Bug Reports for Android Applications. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 673–686. https://doi.org/10.1145/2786805.2786857

[21] Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. 2015. Auto-Completing Bug Reports for Android Applications. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. Association for Computing Machinery, New York, NY, USA, 673–686. https://doi.org/10.1145/2786805.2786857

[22] Kevin Moran, Cody Watson, John Hoskins, George Purnell, and Denys Poshyvanyk. 2018. Detecting and Summarizing GUI Changes in Evolving Mobile Apps. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 543–553. https://doi.org/10.1145/3238147.3238203

[23] Brad A. Myers and Mary Beth Rosson. 1992. Survey on User Interface Programming. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Monterey, California, USA) *(CHI '92)*. Association for Computing Machinery, New York, NY, USA, 195–202. https://doi.org/10.1145/142750.142789

[24] Jakob Nielsen and Rolf Molich. 1990. Heuristic Evaluation of User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Seattle, Washington, USA) *(CHI '90)*. Association for Computing Machinery, New York, NY, USA, 249–256. https://doi.org/10.1145/97243.97281

[25] Donald A. Norman and Stephen W. Draper. 1986. *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., USA.

[26] Ian Sommerville. 2015. *Software Engineering* (10th ed.). Pearson.

[27] Amanda Swearngin, Mira Dontcheva, Wilmot Li, Joel Brandt, Morgan Dixon, and Andrew J. Ko. 2018. *Rewire: Interface Design Assistance from Examples*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3173574.3174078

[28] Dehai Zhao, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020. Seenomaly: Vision-Based Linting of GUI Animation Effects against Design-Don't Guidelines. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 1286–1297. https://doi.org/10.1145/3377811.3380411