A Survey on the Use of Computer Vision to Improve Software Engineering Tasks

Mohammad Bajammal, Andrea Stocco, *Member, IEEE Computer Society*, Davood Mazinanian and Ali Mesbah, *Member, IEEE Computer Society*

Abstract—Software engineering (SE) research has traditionally revolved around engineering the source code. However, novel approaches that analyze software through computer vision have been increasingly adopted in SE. These approaches allow analyzing the software from a different complementary perspective other than the source code, and they are used to either complement existing source code-based methods, or to overcome their limitations. The goal of this manuscript is to survey the use of computer vision techniques in SE with the aim of assessing their potential in advancing the field of SE research. We examined an extensive body of literature from top-tier SE venues, as well as venues from closely related fields (machine learning, computer vision, and human-computer interaction). Our inclusion criteria targeted papers applying computer vision techniques that address problems related to any area of SE. We collected an initial pool of 2,716 papers, from which we obtained 66 final relevant papers covering a variety of SE areas. We analyzed what computer vision techniques have been adopted or designed, for what reasons, how they are used, what benefits they provide, and how they are evaluated. Our findings highlight that visual approaches have been adopted in a wide variety of SE tasks, predominantly for effectively tackling software analysis and testing challenges in the web and mobile domains. The results also show a rapid growth trend of the use of computer vision techniques in SE research.

Index Terms—Computer Vision, Software Engineering, Survey.

1 Introduction

All areas of the software engineering (SE) lifecycle—such as requirements, design, development, and testing— often have the ultimate goal of contributing to a fundamental product of software engineering: the source code. Accordingly, a wide range of SE activities have typically revolved around the source code, whether to improve its quality, reliability, maintainability, or increase developers' productivity. A relatively more recent—and less explored—alternative approach to SE is the adoption of a *computer vision perspective*. This approach utilizes one or more computer vision (CV) algorithms to extract, analyze, or process visual aspects pertaining to the software. The objective is still focused on solving a SE problem or task, but using visual techniques instead of relying merely on the source code. As an example, a typical CV approach might involve screenshot image comparison to compare or analyze two graphical user interfaces (GUI) for testing purposes.

CV approaches have yielded promising results in developing robust and accurate solutions for various SE tasks. For instance, they have been successfully adopted to improve regression testing of GUIs [1, 2, 3], to identify cross-browser incompatibilities in web pages [4, 5, 6], to perform bug detection and automated program repair [7, 8], or to simplify software requirements modelling [9, 10].

In this manuscript, we survey the literature on the use of computer vision in performing software engineering tasks. The aim of this work is to explore and analyze the adoption of CV approaches in the field of software engineering. Our work highlights new techniques and perspectives of addressing existing research topics in SE, what benefits they may provide compared to existing approaches, and what limitations they might bear. We believe this can be helpful in providing a distilled and concise overview of CV approaches in software engineering, building a concrete understanding of the advances made, and synthesizing insights for future research directions.

We conducted the survey by formulating a number of research questions to fulfill the goal of the study; we then proceeded by systematically collecting a pool of publications, and applied a number of inclusion and exclusion criteria. Subsequently, we analyzed and synthesized the collected papers by taking into account a number of dimensions, such as what area of software engineering (e.g., testing, maintenance) they brought benefit to, what specific task is being addressed (e.g., regression testing), what CV techniques have been used, and what is the rationale for their adoption. This manuscript makes the following contributions:

- the first survey on the use of computer vision in software engineering tasks.
- a study of the software engineering areas and tasks benefiting from CV approaches.
- a synthesis of the motivations behind the use of CV approaches in software engineering research.
- an analysis of the CV methods used in software engineering as well as their benefits.
- an investigation of the open challenges pertaining to the use of CV approaches in software engineering.

- *Mohammad Bajammal, Davood Mazinanian, and Ali Mesbah are with the University of British Columbia, Vancouver, BC, Canada.*
- *Andrea Stocco is with the Università della Svizzera Italiana, Switzerland.*

2 Prior Work

To the best of our knowledge, there are no existing surveys or systematic literature reviews that share a similar goal as this manuscript. Accordingly, this section discusses a broader range of research areas in order to put this manuscript in its proper context. We therefore begin by discussing secondary studies concerning CV-based techniques in various (non-SE) engineering fields. Second, we explore other surveys that are interdisciplinary in nature, given the interdisciplinary nature of our survey. Finally, we discuss visual GUI testing techniques, an area with a relatively large number of papers employing CV techniques.

Surveys on Computer Vision-based Engineering. In this subsection, we discuss some of the surveys or systematic literature reviews concerning the use of CV in various engineering fields. We note that these are non-SE engineering fields (e.g., aerospace or automotive engineering) and are included here for the sake of completeness.

Kumar [11] catalogues the fabric defect detection methodologies reported in about 150 references into three main categories: statistical, spectral and model-based. They conclude that despite the significant progress in last decade, the problem of fabric defect detection still remains challenging and requires effort by combining existing approaches. Kanellakis and Nikolakopoulos [12] present a comprehensive literature review on vision based applications for unmanned aerial vehicles (UAVs) focusing mainly on current developments and trends. CV techniques are used mainly for visual localization and mapping, obstacle detection and avoidance, aerial target tracking, and guidance. Among the limitations, it is mentioned that CV algorithms are based on rigid assumptions such as low speed vehicles that do not account for fast scene alterations. Thus, the main challenge is to design solutions that can quickly react to ever changing sceneries, characterized by a high degree of dynamism and evolution. Liu and Dai [13] discusse the CV solutions for UAVs from three main families, namely visual navigation, aerial surveillance and airborne visual Simultaneous Localization and Mapping (SLAM). Al-Kaff et al. [14] provide another survey of CV techniques for UAVs, particularly visual navigation algorithms, obstacle detection and avoidance and aerial decision-making. It is mentioned that artificial perception applications have represented important advances in the latest years in the expert system field related to unmanned aerial vehicles.

Gandhi and Triveli [15] discuss the recent research on pedestrian detection and collision prediction. Among the information gathered by the various sensors, the camera's image is one of the most used, along with visual analysis techniques for behaviour modelling in accident prediction, direction estimation, and collision prediction. Brunetti et al. [16] discuss vision-based pedestrian detection systems pertaining to three different application fields: video surveillance, human-machine interaction and analysis. Notably, they discuss both the differences between 2D and 3D vision systems, and indoor and outdoor systems. Janai et al. [17] provides a comprehensive survey on problems, datasets, and methods in computer vision for autonomous vehicles. First, they overview the datasets and benchmarks used in autonomous driving research. Then, the discuss the state of the art on several relevant topics, including recognition, reconstruction, motion estimation, tracking, scene understanding, and end-to-end learning.

In contrast, this manuscript aims at surveying the use of CV in the requirements, design, development, testing, or maintenance of software.

Interdisciplinary Surveys in SE. Interdisciplinary surveys are often used to collect and analyze a body of knowledge across the boundaries between two or more fields. Here, we discuss some of the surveys or systematic literature reviews that have analyzed scientific and social fields from a software engineering perspective.

Zhang et al. [18] provide a comprehensive survey of techniques for testing machine learning systems (ML testing). The survey covers 144 papers on different testing properties such correctness, robustness, and fairness, testing components (e.g., data, learning program, and frameworks), testing workflow (e.g., test generation and test evaluation), and application scenarios (e.g., autonomous driving, machine translation). The paper also analyses trends concerning datasets, research trends, and research focus, concluding with research challenges and promising research directions in ML testing. Beszédes [19] performed a systematic analysis of fault localization literature across different engineering fields, with the aim to find solutions in non-software areas that could be successfully adapted to software fault localization. Among their findings, some classes of methods in computer networks literature are good candidates for adaptation, and could potentially be reused for software fault localization. Van der Linden and Hadar [20] performed a systematic literature review of physics of notation applications, a conceptual modelling language used for requirement specification. They analyzed what notations have been evaluated and designed using the physics of notation, for what reasons, to what degree applications consider requirements of their notation's users, and how verifiable these applications are.

Sabaren et al. [21] conduct a systematic literature review of cross-browser regression testing. In their survey, their goal was to collect the various techniques that have been proposed to perform cross-browser testing. The authors also describe several challenges in this specific context, such as the automatic identification of dynamic components in a user interface, which undermines the effectiveness of proposed testing techniques, causing many false positives in practice. We note that the survey of Sabaren et al. [21] have found 11 papers that happened to be in our final pool of 66 collected papers. This is a happenstance since our survey has a completely different objective. The work by Sabaren et al. [21] answers the following question: what approaches have been used to conduct cross-browser regression testing. In contrast, our work is not concerned at all with that problem. Our work answers the following question: in what ways have computer vision been used to advance software engineering. The reason we had some common papers is because regression testing happened to be an area where visual techniques were found to be particularly useful. However, in terms of the scope and objective, there is no overlap. In other words, Sabaren et al. [21] focus on a specific problem (i.e., cross-browser regression testing), regardless of what approaches were used (e.g., DOM analysis, state space navigation, visual analysis). That is, the survey in Sabaren et al. [21] is *problem-specific* but approach-agnostic. In contrast, our survey is *approach-specific* but problem-agnostic. We focus on a specific approach (i.e., computer vision techniques), but consider its potential for any area of software engineering (e.g., testing, maintenance, development, design, requirements).

In summary, none of the aforementioned surveys have a similar goal as that of this manuscript, which is to examine the use of CV techniques in software engineering tasks.

Visualization Research. Visualization is the process of creating diagrams, charts, or any other kind of representation, from a given dataset. Visualization is part of any scientific process regardless of the field, and therefore has also been used in software engineering. There are a number of surveys on the use of visualization in various aspects of software engineering, such as surveys on visualization for software security [22, 23], surveys on visualization for static analysis [24], development coordination [25], maintenance and evolution [26, 27], to name a few. Visualization, however, is not the scope of this survey.

Visual GUI Testing. Issa et al [28] first introduced the notion of *visual testing* as a subset of traditional GUI testing. In their analysis, the authors conducted a study of bugs in four open source systems, and found that visual bugs represent between 16% and 33% of reported defects in those systems. In recent years, researchers and practitioners have started conducting empirical experiments aiming at understanding the comparative performance of a few visual testing approaches. For instance, Alégroth et al. [29, 30] present a case study of the benefits and challenges of using visual GUI testing by the team at one software company. In another study, Alégroth et al. [31] study the applicability and feasibility of Visual GUI testing in an industrial Continuous Integration environment, describing the main challenges faced by researchers to make it effective in practice. Garousi et al. [32] compare two popular visual testing tools (Sikuli and JAutomate) in one industrial project, and go through differences in test creation process, execution, and maintenance.

All such works analyze different technical and social aspects related to the use of Visual GUI testing in a specific context (i.e., the development and maintenance of test code). In contrast, our work is agnostic to any specific area or context. That is, it does not aim to focus on GUI testing. Rather, the goal is to broadly examine the use of visual techniques across any software engineering area (e.g., requirements, design, development, testing, maintenance) and for any task (e.g., refactoring, reverse engineering, regression testing).

3 Methodology

In order to conduct the survey in a thorough and structured manner, we follow the established guidelines by Kitchenham et al. [33]. We begin by introducing terminology and concepts needed to understand the remainder of this manuscript. Next, we define the scope of the work and flesh it out into specific research questions we aim to answer in this survey. We then describe the details of the paper collection process. Finally, we specify the inclusion and exclusion criteria applied to select the most relevant body of work from the existing literature.

3.1 Definitions

In order to categorize the use of computer vision approaches in software engineering, we use the terms *areas* and *tasks*. Software engineering areas are the various stages in the software lifecycle [34]. Examples of SE areas include software requirements, software design, and software testing. Within each area, different *tasks* can be defined. Each task is a specific activity that aims to achieve a well-defined objective related to that area. For instance, we refer to unit testing or regression testing as SE tasks within the software testing SE area, whereas code migration or code refactoring are tasks within software maintenance area. Accordingly, the rationale for using these two terms is to discuss our findings in more precise levels of granularity, in order to be able to analyze the findings across areas and for tasks within a specific area.

Next, we define the following terms in order to clarify which aspect of computer vision is being discussed:

Definition 1 (Visual Artifact). *A visual artifact is any datum that satisfies the following two conditions: (1) it constitutes a digital image or video, and (2) it is associated with one or more software engineering area(s).*

Definition 2 (Visual Approach). *A visual approach is an algorithm designed to solve a software engineering problem, which incorporates a computer vision method as one or more of its steps, and takes as input one or more visual artifacts, and yields an output that is used to achieve a software engineering task.*

The rationale for defining these two terms is to have precision and clarity when describing how computer vision is used to solve a software engineering problem. We use the term *visual approach* to indicate that the approach used to solve an SE problem is visual in nature (i.e., uses computer vision). We use the term *visual artifact* to refer to software artifacts that are visual in nature, to differentiate them from other software artifacts that are non-visual (e.g., log files, requirements documents). The link between the two terms is that visual artifacts are the visual data consumed by a visual approach. Similarly, a visual approach is the algorithm that needs visual artifacts as input.

To clarify all of the aforementioned terms, we give a simple example. Consider the case of cross-browser testing, where the goal is to check whether a given web app is being rendered identically in different browsers. Visual approaches for cross-browser testing often take a screenshot of the app in a set of different browsers, and then visually compare the screenshots. In this case, screenshots are the visual artifacts used or extracted from the software, and image comparison is the visual approach used to solve the SE task of cross-browser testing.

3.2 Scope

The scope of this work is to conduct a survey to help structure, curate, and unify the dispersed literature in this research area, and to analyze how computer vision techniques have been used in software engineering, and what

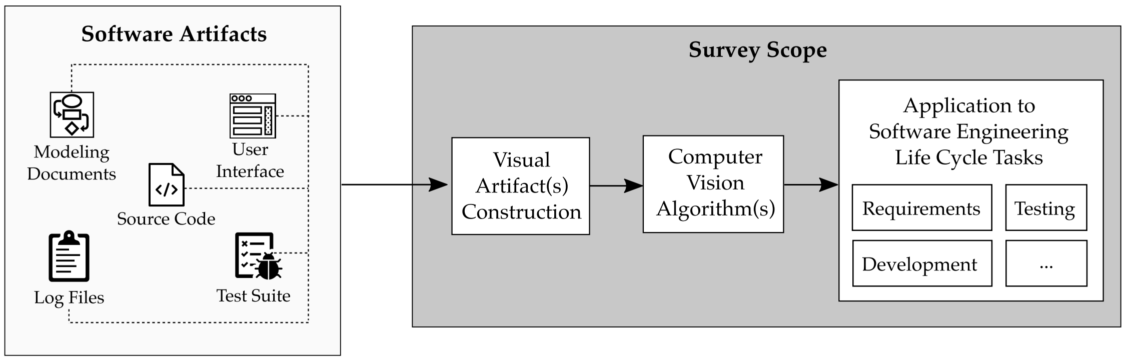


Fig. 1: Overview of the scope of this survey.

are the challenges reported when they were used. This would help shed light on the potential of these techniques, and make them more visible and accessible.

Figure 1 illustrates the scope of this work in relation to the software engineering life cycle and other software artifacts. The figure should be viewed as a multi-step process, beginning with visual artifacts construction, and ending with an application to a software engineering task, as defined in Section 3.1 As shown in the figure, the scope is to survey the following aspects: (1) how are visual artifacts (defined in Section 3.1) constructed or acquired from the software. This aspect is included in the scope because constructing or acquiring visual artifacts is the first step in a computer vision processing pipeline, and therefore discussing the nature of these artifacts and how they are constructed and used should be examined in order to have a well-rounded survey. (2) what computer vision algorithms are used to analyze or process the constructed visual artifacts. This aspect is included in the scope because examining the visual processing or analysis conducted in order to address a given paper's research questions yields insight into how visual techniques can be potentially applied to various tasks of software engineering. (3) what are the software engineering areas and tasks where visual approaches have been used.

The figure also helps clarify what areas are outside the scope of this survey. For instance, the scope is not concerned with works where computer vision techniques were not utilized, or works that do not use any visual artifacts. Section 3.4.1 fleshes out the scope into a detailed set of inclusion and exclusion criteria.

3.3 Research Questions

As discussed in Section 3.2, the scope is to survey the use of computer vision in solving software engineering problems. In this section, we flesh out the scope into the following specific research questions:

RQ1: *What are the main software engineering areas and tasks for which computer vision approaches have been used to date?*

We formulate this RQ in order to construct a high level picture of the areas of software engineering where computer vision approaches were used. This can help identify potential trends of areas with high adoption of computer vision (and, subsequently, investigating why is that the case), and conversely areas where little to no computer vision approaches were used. This can help the software engineering community in identifying potential gaps in the utilization of computer vision for software engineering.

RQ2: *Why are computer vision approaches adopted?*

We formulate this RQ in order to identify common rationales for using computer vision to solve software engineering problems. This understanding of why computer vision approaches were used can subsequently help identify new software engineering areas or tasks where similar problems and rationales exist and therefore potentially benefiting from computer vision approaches.

RQ3: *How are computer vision approaches applied to software and its visual artifacts?*

This RQ is a natural progression of the previous RQs. The previous RQs identified the rationales of using computer vision and the SE areas where computer vision were used. This RQ examines the "how." That is, the mechanism(s) by which computer vision was applied to software. This can help guide the implementation of computer vision approaches to solve software engineering problems.

RQ4: *How are software engineering tasks that leverage computer vision techniques evaluated?*

This RQ examines the methods used to evaluate the use of computer vision approaches in software engineering problems, as well as a summary of the reported limitations and challenges. This may help with selecting an evaluation strategy when exploring the use of a computer vision approach, and informing adopters of potential challenges.

3.4 Paper Collection

Figure 2 shows our paper search and selection process. In order to collect as many relevant literature as possible, we used two types of sources for paper collection: paper repository databases, and major software engineering venues.

Paper Repository Databases. To conduct our search, we used the databases of the following well-known publishers of scientific literature: IEEE Xplore, ACM Digital Library,

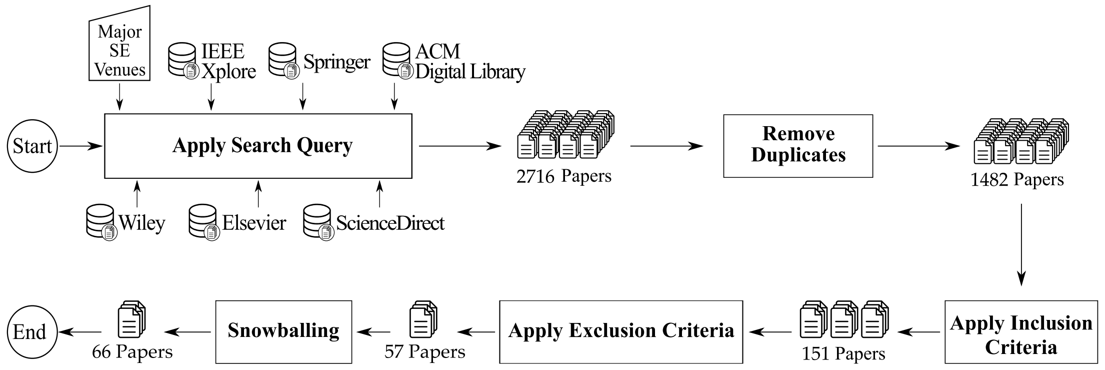


Fig. 2: Overview of the paper collection process.

ScienceDirect, Springer, Wiley, and Elsevier. The search covers papers that have been published until June 2020 (the date of this writing). We used more than one database to ensure collecting as many papers as possible from all known publishers.

Software Engineering Venues. The preceding selection of paper repositories aims at casting a wide net in order to capture as many relevant literature as possible. However, since the databases contain an extremely large number of papers, it is possible that papers relevant to our survey are lost in the vast number of returned papers.

For this reason, and in order to make sure we collect highly relevant papers, we complemented the database search with a manual issue-by-issue search within the conference proceedings and journal articles from top-tier software engineering venues (listed in Table 1). The final pool of collected papers is the combined list of papers from both the database search and the SE venues search.

Interdisciplinary Venues. Given the interdisciplinary nature of this work, we also performed manual issue-by-issue search within the conference proceedings of relevant fields, namely, computer-human interaction, computer vision and machine learning. We selected the top three venues (based on the h5 index from Google Scholar) from each field. The searched venues are listed in the last section of Table 1 (under "Interdisciplinary Venues").

Search Query. For each of the aforementioned sources, we performed a search query using various combinations of terms to retrieve papers in different software engineering areas that are potentially using computer vision. The query was performed on all data fields of the paper, returning matches to either the *title*, *abstract*, *keywords*, or *text* of the paper. The query is composed of two parts: keywords related to the approach and keywords of the various software engineering areas. Keywords of the approach include strings such as "computer vision" or "visual". They were included in the query in order to indicate our interest in works that use computer vision or visual approach. Keywords describing the areas include strings such as "development" or "testing" or any of the software engineering life cycle phases (e.g., requirements, maintenance). The final applied search query is as follows:

[computer vision ***OR*** *image processing* ***OR*** *image analysis* ***OR*** *visual]* ***AND*** *[requirements* ***OR*** *design* ***OR*** *development* ***OR*** *testing* ***OR*** *maintenance* ***OR*** *comprehension]*

The result of this query led to an initial pool of 2,716 papers, which was further filtered in the next stages.

Duplicates removal. During the paper collection step, we aimed to be thorough by including as many paper sources as possible in order to capture all potentially relevant works. However, this resulted in many duplicate papers since a given paper might be included in more than one database and venue. Therefore, we filtered the collected pool of papers by removing duplicate works based on their titles.

3.4.1 Inclusion and Exclusion Criteria

The search conducted on the databases and venues is, by design, very inclusive. This allows us to collect as many papers as possible in our pool. However, this generous inclusivity results in having papers that are not directly related to the scope of this survey. Accordingly, we define a set of specific inclusion and exclusion criteria and apply them to each paper in the pool, and remove papers not meeting the criteria. This ensures that each collected paper is inline with our scope and research questions.

Inclusion criteria. We define the following inclusion criteria: (1) The paper should be contributing to any stage of the software engineering process, whether in early requirements and modeling, through development and design, or finally testing and maintenance. We included this criteria in order to focus on software engineering papers. This is because we found a notable number of computer vision papers that were in fields other than software engineering (e.g., biology). (2) The paper should include a computer vision processing of the software or its artifacts. That is, the work achieves its objective (whether partially or fully) by extracting, analyzing, or processing visual artifacts pertaining or relevant to the software. This is an important and key criterion for paper selection because it ensures we meet the core scope of our manuscript (i.e., surveying the use of computer vision in software engineering). (3) The paper should be a full technical research paper that has a detailed description of the visual approach utilized. This criterion is imposed in order to have sufficient information to answer our research questions. Answering our research questions, such as RQ3 and RQ4, requires that we examine technical software engineering research papers, as opposed to, for instance, technical magazine articles, industrial white papers, or similar grey literature which do not have sufficient level of detail. Some demo/short papers can be allowed if they have dedicated and detailed sections discussing the detailed mechanism of the visual approach and its evaluation. This enables creating a pool of papers that have rich and detailed information and findings, in order to answer key research questions related to the details of the visual approach, the process of creating visual artifacts, and evaluation strategies. (4) The paper should contain a section dedicated to illustrating some form of quantitative or qualitative evaluation of the technique, or an illustration of its use case. This criterion was imposed in order to enable us to fully answer and explore our research questions regarding evaluation strategies.

These criteria were applied in a group review process by the authors. For each paper in the pool, each author initially checked the title and abstract, and briefly examined the proposed approach and results to ensure that it meets the inclusion criteria. If this check was not sufficient to conclusively decide whether the paper should be included in the pool, we proceeded with a secondary in-depth examination of the paper's objective, methodology, and evaluation to ensure that the inclusion criteria were met. Finally, a discussion among the authors was triggered, to decide on the inclusion of the paper in the final list of works.

Exclusion criteria. During our initial experimental test rounds of paper searching, we observed that a relatively large number of retrieved papers were on visualization research. This is understandable and expected because our queries include terms such as image, visual, and design.

Accordingly, we exclude papers published in the area of software visualization for the following reasons. First, the visualization class of algorithms does not constitute a *visual approach*, as defined in Section 3.1. Visualizations do not use any visual artifact of the software *as an input*. Rather, such work perform a final visual output or visual representation of a complete, non-visual, approach. Accordingly, this area of research is excluded since it would be outside the scope of this work. We recall that the scope is to survey visual approaches which, by definition, *consume* visual artifacts pertaining to the software during the course of running their algorithm or processing. Second, in addition to visualization being outside the scope of this survey, it is already a well-known and common aspect of software engineering, and plenty of surveys already exist on the use of visualization in various aspects of software engineering, as mentioned in Section 2.

We also exclude commercial software services or open-source tools that have no corresponding publication, due to the following reasons. First, including services or tools that are not peer-reviewed would negatively impact our ability to conduct the survey because tools and services that are not backed by a publication do not include a detailed explanation of their approach. This reduces our ability to answer key research questions for this survey, such as what specific computer vision techniques were used, what is the visual artifacts construction process, and how were the computer vision algorithms applied to the visual artifacts. Services and tools without a corresponding publication also do not have a thorough systematic evaluation, and therefore we are unable to answer research questions related to how computer vision techniques were evaluated, what are the main findings, and what were the challenges.

TABLE 1: Conference proceedings and journals considered for paper collection (in addition to database search).

	Acronym	Venue
SE Conferences	ICSE	International Conference on Software Engineering
	FSE	International Symposium on Foundations of Software Engineering
	ESEC/FSE	Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering
	ASE	International Conference on Automated Software Engineering
	ESEM	International Symposium on Empirical Software Engineering and Measurement
	ICST	International Conference on Software Testing, Verification and Validation
	ISSTA	International Symposium on Software Testing and Analysis
	MSR	International Conference on Mining Software Repositories
	RE	International Requirements Engineering Conference
	ICSME	International Conference on Software Maintenance and Evolution
	MODELS	International Conference on Model Driven Engineering Languages and Systems
	ISSRE	International Symposium on Software Reliability Engineering
	EASE	Evaluation and Assessment in Software Engineering
SE Journals	TSE	Transactions on Software Engineering
	EMSE	Empirical Software Engineering
	TOSEM	Transactions on Software Engineering and Methodology
	JSS	Journal of Systems and Software
	JSEP	Journal of Software: Evolution and Process
	STVR	Software Testing, Verification and Reliability
	ASE	Automated Software Engineering
	IEEE SOFTWARE	IEEE Software
	IET SOFTW.	IET Software
	IST	Information and Software Technology
	SQJ	Software Quality Journal
Interdisciplinary Venues	CHI	Conference on Human Factors in Computing Systems
	CSCW	Conference on Computer-Supported Cooperative Work
	UbiComp	Conference on Pervasive and Ubiquitous Computing
	UIST	Symposium on User Interface Software and Technology
	NeurIPS	Conference on Neural Information Processing Systems
	ICLR	International Conference on Learning Representations
	ICML	International Conference on Machine Learning
	CVPR	Conference on Computer Vision and Pattern Recognition
	ECCV	European Conference on Computer Vision
	ICCV	International Conference on Computer Vision

3.4.2 Snowballing

At the end of searching database repositories and conference proceedings and journals, and applying inclusion and exclusion criteria, we obtained a total of 57 unique papers. Next, to mitigate the risk of omitting relevant literature from this survey, we also performed backward snowballing [35] by inspecting the references cited by the collected papers so far. Nine additional papers were retrieved during this phase, which led to a final pool of 66 unique papers. Table 3 shows the final pool of papers that will be discussed and analyzed in the remainder of this work.

TABLE 2: Collected data items (Synthesis Matrix)

Field	Use
Title	Documentation
Author(s)	Documentation
DOI identification number	Documentation
Abstract	Paper Selection
Text	Paper Selection
Venue	RQ1
Year	RQ1
Target platform	RQ1
Software engineering area	RQ1
Software engineering task	RQ1
Reasons for adopting CV approach	RQ2
Visual artifact(s) used	RQ3
CV algorithm(s) used	RQ3
Evaluation process & challenges	RQ4
Main results	RQ4
Limitations of CV methods used	RQ4

3.4.3 *Extracted Information*

For each retrieved paper, we collect a set of data necessary to answer the research questions. Table 2 shows the list of data collected from each paper and their mapping to each research question. As shown in the table, the title, author(s), and document ID were used for documentation purposes to keep track of the various papers. The abstract and text were used for the paper selection process and applying the inclusion and exclusion criteria. The venue, year, software engineering area and task of each paper was also collected in order to discuss and answer RQ1. We also extract the target platform for each paper, which is the type of computing device (e.g., desktop, mobile) that the analyzed software runs on. A list of reasons for adopting computer vision was also extracted from each paper in order to answer RQ2. The visual artifact(s) and the visual approach utilized were also identified in each paper in order to discuss RQ3. Finally, we log the evaluation process and the results and findings from each collected paper in order to answer RQ4. All these data are collected, analyzed, and used to synthesize the findings for the rest of this manuscript. In order to facilitate the use of this data by the general research community, the data have been made publicly available at http://tiny.cc/tse-2020.

4 Findings

4.1 Trends and Landscape

At the end of the paper collection process, we obtained a pool of papers spanning the years 2001-2020 (June 2020). Table 3 shows the final list of 66 papers. Figure 3 shows the distribution of the retrieved pool of papers across different years of publication. Overall, we observed a generally increasing trend in the use of computer vision approaches in software engineering. This research area is also relatively new, with more than half of the papers in our pool published in the past five years. Furthermore, Figure 4 depicts the cumulative number of publications per software engineering area across different years. The results indicate that software testing is the area exhibiting the most rapid increase in terms of the number of publications wherein a computer vision technique is utilized.

Figure 5 shows the distribution of the published papers across venues. The main venues in which computer vision approaches for software engineering were published are the Conference on Human Factors in Computing Systems (CHI) with 11 papers, the International Conference on Software Engineering (ICSE) with nine papers, and the International Conference on Automated Software Engineering (ASE) with eight papers. The presence of traditional SE venues as well as venues from other fields (e.g. CHI) in Figure 5 provides some indication that research on the use of computer vision for software engineering tasks is an interdisciplinary field.

4.2 Areas, Tasks, and Platforms (RQ1)

To study the usefulness of visual techniques for SE, we analyzed the selected papers to find out which *SE areas* have been explored, for which *tasks*, and on which *platforms* they were used. As defined in section 3.1, SE areas are high-level stages of the software engineering life cycle, such as requirements, testing, or development. SE tasks are more fine-grained activities, such as unit testing or regression testing. The platforms are the types of computing devices (e.g., desktop, mobile) that the analyzed software runs on. We further looked into the papers' discussion sections to gain insights from the authors about other areas in which the proposed technique could potentially be applied.

4.2.1 *Software Engineering Areas and Tasks*

Figure 6 presents the papers distribution across different SE areas and tasks. Note that the number of papers indicated in the figure is more than the total number of papers in the pool. This is due to the fact that for some papers, the presented approach can be utilized for more than one task. We now discuss more in detail the trends of Figure 6.

Testing. Software testing is the most common research area for which approaches using computer vision are proposed, accounting for approximately half of all collected papers. A closer look at the publications in this area reveals interesting trends. Most of the studies use visual methods to facilitate *acceptance* and *regression testing* e.g., by comparing visual artifacts (e.g., the GUIs) with each other or with respect to a given oracle. Without adopting computer vision, developers

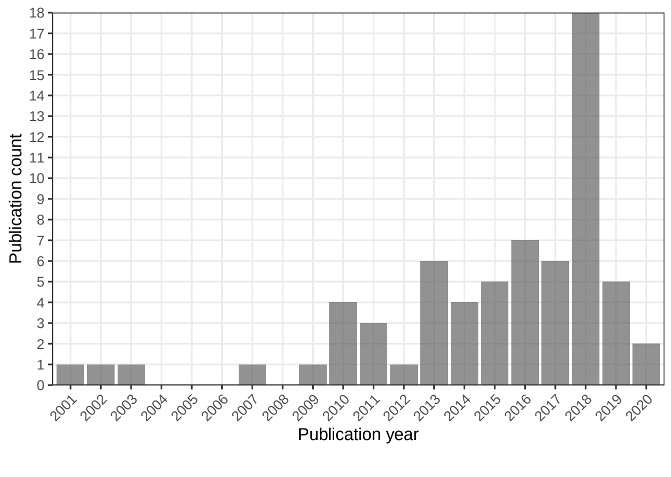


Fig. 3: Distribution of publications across years.

TABLE 3: Collected pool of papers (in chronological order).

Reference	Title	Venue	Year
Landay and Myers [36]	Sketching Interfaces: Toward More Human Interface Design	IEEE Computer	2001
Caetano et al. [37]	JavaSketchIt: Issues in Sketching The Look of User Interfaces	AAAI	2002
Fails and Olsen [38]	A Design Tool for Camera-based Interaction	CHI	2003
Coyette et al. [39]	Multi-fidelity Prototyping of User Interfaces	INTERACT	2007
Zheng et al. [40]	Correlating Low-level Image Statistics with Users - Rapid Aesthetic and Affective Judgments of Web Pages	CHI	2009
Chang et al. [1]	GUI Testing Using Computer Vision	CHI	2010
Choudhary et al. [41]	WEBDIFF: Automated Identification of Cross-browser Issues in Web Applications	ICSM	2010
Li et al. [9]	FrameWire: A Tool for Automatically Extracting Interaction Logic from Paper Prototyping Tests	CHI	2010
Dixon and Fogarty [42]	Prefab: Implementing Advanced Behaviors using Pixel-based Reverse Engineering of Interface Structure	CHI	2010
Delamaro et al. [43]	Using Concepts of Content-based Image Retrieval to Implement Graphical Testing Oracles	STVR	2011
Dixon et al. [44]	Content and Hierarchy in Pixel-based Methods for Reverse Engineering Interface Structure	CHI	2011
Seifert et al. [45]	Mobidev: A Tool for Creating Apps on Mobile Phones	MobileHCI	2011
Choudhary et al. [46]	Crosscheck: Combining Crawling and Differencing to Better Detect Cross-browser Incompatibilities in Web Applications	ICST	2012
Givens et al. [47]	Exploring The Internal State of User Interfaces by Combining Computer Vision Techniques with Grammatical Inference	ICSE	2013
Liang et al. [48]	SeeSS: Seeing What I Broke – Visualizing Change Impact of Cascading Style Sheets (CSS)	UIST	2013
Scharf and Amma [10]	Dynamic Injection of Sketching Features Into GEF-based Diagram Editors	ICSE	2013
Alégroth et al. [2]	JAutomate: A Tool for System- and Acceptance-test Automation	ICST	2013
Semenenko et al. [4]	Browserbite: Accurate Cross-Browser Testing via Machine Learning over Image Features	ICSM	2013
Roy Choudhary et al. [5]	X-PERT: Accurate Identification of Cross-Browser Issues in Web Applications	ICSE	2013
Lin et al. [3]	On the Accuracy, Efficiency, and Reusability of Automated Test Oracles for Android Devices	TSE	2014
Mahajan and Halfond [7]	Finding HTML Presentation Failures Using Image Comparison Techniques	ASE	2014
Amalfitano et al. [49]	Towards Automatic Model-in-the-loop Testing of Electronic Vehicle Information Centers	WISE	2014
Selay et al. [6]	Adaptive Random Testing for Image Comparison in Regression Web Testing	DICTA	2014
Bao et al. [50]	scvRipper: Video Scraping Tool for Modeling Developers' Behavior Using Interaction Data	ICSE	2015
Nguyen and Csallner [51]	Reverse Engineering Mobile Application User Interfaces with REMAUI	ASE	2015
Burg et al. [52]	Explaining Visual Changes in Web Interfaces	UIST	2015
Mahajan and Halfond [53]	Detection and Localization of HTML Presentation Failures Using Computer Vision-Based Techniques	ICST	2015
Hori et al. [54]	An Oracle based on Image Comparison for Regression Testing of Web Applications	SEKE	2015
Reinecke et al. [55]	Enabling Designers to Foresee Which Colors Users Cannot See	CHI	2016
Deka et al. [56]	ERICA: Interaction Mining Mobile Apps	UIST	2016
Ponzanelli et al. [57]	Too Long; Didn't Watch! Extracting Relevant Fragments from Software Development Video Tutorials	ICSE	2016
Mahajan et al. [58]	Using Visual Symptoms for Debugging Presentation Failures in Web Applications	ICST	2016
Feng et al. [59]	Multi-objective Test Report Prioritization Using Image Understanding	ASE	2016
Patrick et al. [60]	Automatic Test Image Generation Using Procedural Noise	ASE	2016
He et al. [61]	X-Check: A Novel Cross-browser Testing Service Based on Record/Replay	ICWS	2016
Deka et al. [62]	Rico: A Mobile App Dataset for Building Data-Driven Design Applications	UIST	2017
Wan et al. [63]	Detecting Display Energy Hotspots in Android Apps	STVR	2017
Bao et al. [64]	Extracting and Analyzing Time-series HCI Data from Screen-captured Task Videos	EMSE	2017
Zhang et al. [65]	Sketch-guided GUI Test Generation for Mobile Applications	ASE	2017
Chen et al. [66]	UI X-Ray: Interactive Mobile UI Testing Based on Computer Vision	IUI	2017
Wu et al. [67]	Automatic Alt-text: Computer-generated Image Descriptions for Blind Users on a Social Network Service	CSCW	2017
Reiss and Miao [68]	Seeking the User Interface	ASE J.	2018
Kıraç et al. [69]	VISOR: A Fast Image Processing Pipeline with Scaling and Translation Invariance for Test Oracle Automation of Visual Output Systems	JSS	2018
Leotta et al. [70]	Pesto: Automated Migration of DOM-based Web Tests Towards the Visual Approach	STVR	2018
Bajammal and Mesbah [71]	Web Canvas Testing through Visual Inference	ICST	2018
Xu and Miller [72]	Cross-Browser Differences Detection Based on an Empirical Metric for Web Page Visual Similarity	TOIT	2018
Kuchta et al. [73]	On the Correctness of Electronic Documents: Studying, Finding, and Localizing Inconsistency Bugs in PDF Readers and Files	EMSE	2018
Bao et al. [74]	VT-Revolution: Interactive Programming Video Tutorial Authoring and Watching System	TSE	2018
Moran et al. [75]	Automated Reporting of GUI Design Violations for Mobile Apps	ICSE	2018
Chen et al. [76]	From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation	ICSE	2018
Sun et al. [77]	Neural Program Synthesis from Diverse Demonstration Videos	ICML	2018
Lim et al. [78]	Ply: A Visual Web Inspector for Learning from Professional Webpages	UIST	2018
Moran et al. [79]	Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps	TSE	2018
Stocco et al. [8]	Visual Web Test Repair	FSE	2018
Tanno and Adachi [80]	Support for Finding Presentation Failures by Using Computer Vision Techniques	ICST	2018
Bajammal et al. [81]	Generating Reusable Web Components from Mockups	ASE	2018
Moran et al. [82]	Detecting and Summarizing GUI Changes in Evolving Mobile Apps	ASE	2018
Natarajan and Csallner [83]	P2A: A Tool for Converting Pixels to Animated Mobile Application User Interfaces	MOBILESoft	2018
Osman et al. [84]	An Automated Approach for Classifying Reverse-engineered and Forward-engineered UML Class Diagrams	SEAA	2018
Xiao et al. [85]	Automatic Identification of Sensitive UI Widgets based on Icon Classification for Android Apps	ICSE	2019
Huang et al. [86]	Swire: Sketch-based User Interface Retrieval	CHI	2019
Zhao et al. [87]	ActionNet: Vision-Based Workflow Action Recognition from Programming Screencasts	ICSE	2019
Yu et al. [88]	LIRAT: Layout and Image Recognition Driving Automated Mobile Testing of Cross-Platform	ASE	2019
Swearngin and Li [89]	Modeling Mobile Interface Tappability using Crowdsourcing and Deep Learning	CHI	2019
Yuan and Li [90]	Modeling Human Visual Search Performance on Realistic Webpages using Analytical and Deep Learning Methods	CHI	2020
Wu et al. [91]	Predicting and Diagnosing User Engagement with Mobile UI Animation via a Data-Driven Approach	CHI	2020

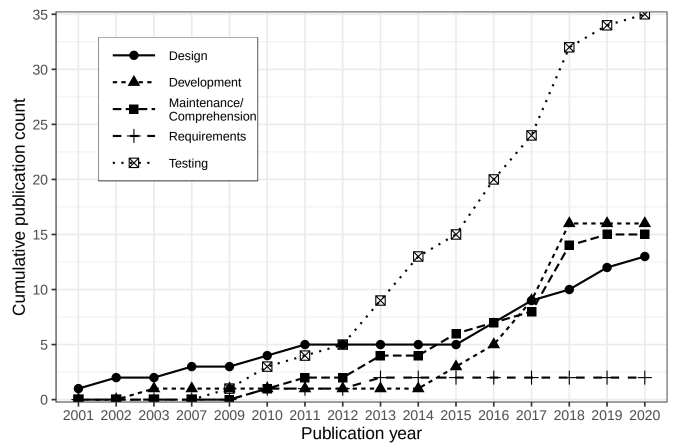


Fig. 4: Cumulative distribution of publications across years per SE area. Testing is the most common area, followed by development and maintenance.

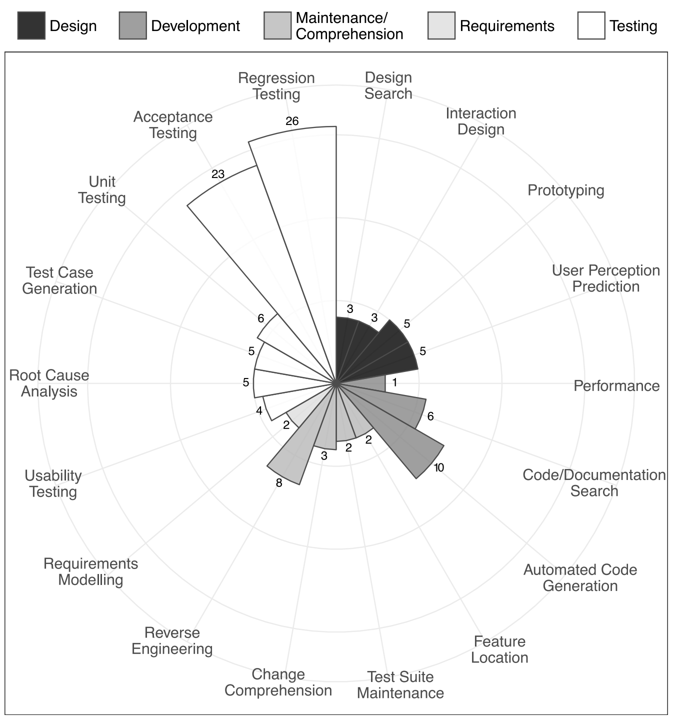


Fig. 6: Papers distribution across different Software Engineering Areas and Tasks

would most likely need to perform some kind of manual evaluation, e.g., through eyeball analysis to spot deviations from the expected visual presentation—a daunting and error-prone task.

Apart from GUI comparisons, CV techniques have been also utilized for other software testing tasks. For example, Kuchta et al. [73] introduce a technique for regression testing of PDF reader software and localizing faulty parts of PDF files. The adopted technique exploits *differential testing*, where the (visual) output of multiple implementations of a program—the PDF viewer—is compared to the same input to spot deviations. In another work, Leotta et al. [70] propose an automated code migration tool for automatically converting end-to-end Selenium web tests to visual web tests based on Sikuli's [1] image recognition capabilities. Kıraç et al. [69] provide an image comparison technique for black-box, regression testing of visual-output software used in consumer electronics. The approach removes noise to eliminate image differences caused by scaling and translation, and is evaluated on the output of digital TVs. Bajammal and Mesbah [71] propose a technique to test state-free canvas elements on web pages by reverse-engineering a visual model. This allows unit testing of the specific visual elements contained in the canvas.

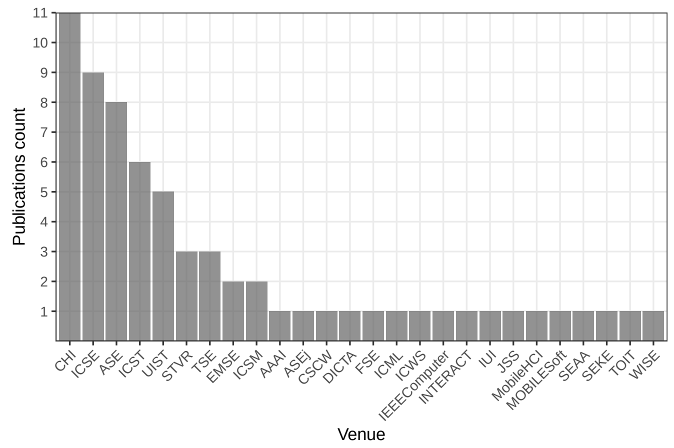


Fig. 5: Distribution of the publications across venues.

A significant class of software testing techniques leveraging visual methods aims at automatically identifying *cross-browser incompatibilities* (XBIs) for web applications [4, 5, 6, 41, 46, 61, 72]. XBIs are frequently-occurring issues in web pages' appearance and/or behavior when the same page is viewed on different web browsers [5]. Identifying such differences requires laborious human judgement, which can be effectively reduced using an automated visual-based technique. A recent literature review by Sabaren et al. [21] surveys the techniques proposed to tackle XBIs.

A similar problem involves using visual methods for *root cause analysis* of presentational issues occurring on web pages [7, 53, 58]. In addition to the web domain, root cause analysis is also used to identify rendering issues in PDF files [73], as well as, the causes of excessive energy consumption of UIs in mobile applications [63].

Several approaches attempt to automate test execution using visual techniques. An example is *record-and-replay testing* where the screenshots of the GUI and the human tester's actions and inputs performed on the GUI are recorded [1, 2, 3, 61] . Popular visual record-and-replay tools are Sikuli [1] and JAutomate [2], which allow fast and easy replay of the same sequence of actions: the tool conducts a visual search on the current visible contents of the screen to detect the widgets' locations, triggers the recorded actions and inputs, and finally performs a visual assertion, comparing the observed visual outcome with the expected oracle.

Besides record-and-replay tools, test automation with visual analysis has been successfully applied to *automotive software engineering*. Amalfitano et al. [49] propose a tool to automate the testing of the emulated vehicle information systems' panels. The testers can locate visual elements on the panel and specify their properties; the tool allows to check the panel's output with respect to these properties at pre-defined timestamps.

Zheng et al. [40], Yuan and Li [90], Deka et al. [56], and Deka et al. [62] aim to automate the testing of *aesthetics or usability* of web pages. This line of work involves building computer vision models that can predict whether web pages meet certain aesthetic requirements pertaining to the usability of a page (e.g. visual balance of white space and elements, consistent and simple representation of elements on a web page).

Patrick et al. [60] propose an approach based on systematic image manipulation to *automatically generate test input images* for regression and acceptance testing of an epidemiological simulation software. The software's output on these input images is monitored and unexpected deviations reveal bugs or regressions in the code.

To *prioritize test reports*, Feng et al. [59] use the screenshots provided by users to augment the existing textual test prioritization techniques for mobile applications. Finally, to *automatically generate test cases*, Zhang et al. [65] use a visual-aided approach that identifies strokes that testers draw on screenshots taken from the apps. These sketches are used to define test specifications (e.g., coordinates where a visual object should be positioned to on the screen), which are subsequently used to generate test cases automatically.

Design. Our survey included a number of papers aiming at facilitating the design stage of software systems by means of computer vision. Li et al. [9] propose a tool to help with *prototyping* software designs. Using computer vision techniques, a video recording of hand-drawn GUI design sketches are converted into a digital form. This serves as an interactive, clickable, documentation of the prototype which can be easily shared with stakeholders. The works of Landay and Myers [36], Caetano et al. [37], Coyette et al. [39], and Scharf and Amma [10] also share the same goal of facilitating user interface prototyping by using computer vision to convert hand-drawn GUI sketches to working GUI prototypes.

Deka et al. [56] use a visual technique to learn features from mobile applications' UIs to create a database of UI design samples, forming a benchmark for *design searching*. In a successive work, Deka et al. [62] record the crowed-sourced *interactions* with the application's GUIs. This allows mining these interactions to incorporate them in new designs, and *predicting users' perception* by their interaction with new GUIs. The works of Reinecke et al. [55], Swearngin and Li [89], Wu et al. [91] also propose visual techniques to help UI designers in predicting the user perception of their designs. Reinecke et al. [55] visually examines the color spectrums and arrangements in a web page, and informs UI designers if certain demographics (e.g. color-blind users) would not be able to see certain parts of their designs. Swearngin and Li [89] build a computer vision model that mimics users perception of "tappability" of various elements in a mobile app. Accordingly, if an app designer has an element that is tappable, but would not be perceived as tappable for the average user, the tool would flag such elements. Wu et al. [91] focuses on flagging animations that would be perceived (by the average user) as too fast, chaotic, or lacking transitions. The UI designer is then notified of these issues in order to mitigate them.

Requirements Engineering. Requirements engineering was the least explored area among all collected papers, with only two publications. Visual techniques have been used to generate a digital form of requirement or design models (e.g., UML) by visually processing hand-made sketches [10], or by augmenting existing requirement artifacts to make them user-tractable [9].

Comprehension and Maintenance. Visual techniques have been used in software *reverse engineering*. The REMAUI tool [51] uses computer vision techniques to reverse engineer the UI elements and their hierarchy in a mobile application from a screenshot (or a mockup), which also allows to automatically generate the UI code. Givens et al. [47] performs a similar reverse engineering of the internal state of desktop applications based on visual decomposition of screenshots. Dixon et al. [44] takes this a step further by reverse engineering the hierarchy of interface components. Bajammal and Mesbah [71] reverse engineers the state of web canvas elements from a visual screenshot of the canvas itself, which also enables testing of canvas elements. Deka et. al. [56], [62] captures traces of user's interaction with mobile apps, allowing the mining of user interactions from a large collection of apps.

In another work, Burg et al. [52] use visual techniques for localizing the JavaScript code responsible for the implementation of a single widget that determines an interactive behaviour on a web application (i.e., *feature location*). Lim et al. [78] presents a similar tool but focuses on localizing the CSS implementation responsible for certain visual appearances.

Stocco et al. [8] present a visual approach for *automated test repair*; they propose a technique to repair broken web test cases by visually analyzing test executions. Finally, Leotta et al's approach [70] for migrating Selenium-based web test cases to Sikuli can help in *maintaining* web tests, i.e., when it is required to convert DOM-based locators (e.g., XPath expressions) to modern visual locators.

Development. Visual techniques have been used for *automated code generation*, simplifying the development stage of software engineering. This includes generating UI code from mockups [51, 76, 81], existing mobile apps UI code [56, 62], hand-made sketches [68], or from a video recording depicting the desired behavior [77]. Wu et al. [67] propose a tool that automatically annotates HTML images with suitable alternative texts.

Fails and Olsen [38] present a tool that helps developers in the creation of software that processes live camera feeds, without requiring developers to have computer vision skills. Bao et al. [50] proposes a similar tool that facilitates scraping of developers' screencast videos, which simplifies searching for code and documentation from video tutorials.

Reiss and Miao [68] propose a technique to *search code* from existing repositories based on a given sketch, to make a compilable code from the results. Ponzanelli et al. [57] allow searching relevant code fragments from video tuto-

TABLE 4: Papers distribution across different platforms

Platform	Papers
Web	[4, 5, 6, 7, 8, 9, 40, 41, 46, 48, 52, 53, 54, 58, 61, 67, 70, 71, 72, 78, 81, 90]
Desktop	[1, 2, 10, 36, 37, 38, 39, 42, 43, 44, 47, 50, 55, 57, 59, 60, 64, 68, 73, 74, 77, 84, 87]
Mobile	[3, 45, 51, 51, 56, 62, 63, 65, 66, 75, 76, 79, 80, 82, 83, 85, 86, 88, 89, 91]
Other	[49, 69]

rials using visual techniques. Bajammal et al. [81] generate UI component code (e.g., React, AngularJS) from a visual analysis of a web app's mockup design. Finally, Wan et al. [63] allow to spot energy pitfalls in the UIs of mobile apps, allowing a more performance-aware UI development.

4.2.2 Platforms

Table 4 illustrates the results of our analysis with respect to the platforms in which visual approaches were utilized. More than half of the collected papers target web and mobile platforms.

Web and mobile applications are ubiquitous nowadays, and their sole communication interface with users is through their GUIs. Desktop applications, on the contrary, can often have different interfaces, e.g., a command-line interface, or a network interface where the use of an external client software is required. Hence, it is not surprising for visual approaches to be more utilized in web and mobile domains. However, there are also other interesting platforms [49, 69], (e.g. automotive dashboards or digital TVs) where visual techniques have been successfully applied. This indicates the potential of visual techniques in any platform where software deals with a GUI, or any artifact that is visual in nature.

4.2.3 Summary

This section focused on exploring the areas, tasks, and platforms where computer vision techniques have been proposed to address software engineering problems. We found that software testing is the most common SE area where computer vision techniques have been used. Within the area of software testing, cross-browser compatibility is the most frequent task that uses computer vision. We also found that more than half of the collected papers target web or mobile platforms, as opposed to desktop.

4.3 Rationale (RQ2)

The goal of this RQ is to understand the *motivations* behind the use of visual approaches in the collected publications. For each paper in our pool, we analyzed the paper's full text and noted down the rationales mentioned by the authors for using computer vision to solve the software engineering problem being tackled. This resulted in three main categories, namely, *context-driven, ease of use,* and *robustness* (as will be described in the below). For each paper that did not explicitly mention their rationale, we analyzed the text and classified the paper to the closest rationale category. We now describe the three identified categories of rationale.

4.3.1 Context-driven

Computer vision has been utilized because the context is intrinsically *visual* in nature, which is the case in all the papers that focused on GUIs. Thus, it was natural for the authors to deal with a visual artifact through a computer vision technique. More than half of the selected papers motivate the use of visual approaches as such [1, 2, 6, 9, 10, 43, 49, 51, 52, 53, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 68, 69, 70, 71, 72, 73].

For instance, Chang et al. [1] describe two properties of visual approaches that make them particularly appealing for analyzing GUI-based software: *intuitiveness* and *universality*. They describe how for certain tasks, using visual artifacts— such as a GUI screenshot— is a more spontaneous way of interaction with the software. Due to their graphical nature, elements on the GUI can be most directly represented by screenshots. Non-visual alternatives, such as scripting, would instead require users to manipulate GUI elements through keywords which is arguably less intuitive.

Furthermore, screenshots are easily accessible for all GUI-based applications. Indeed, it is virtually always possible to take a screenshot of a GUI element, across all applications and platforms. This can make it attractive to propose techniques based on analyzing visual artifacts.

4.3.2 Ease of Use

As a second main motivation, researchers have utilized visual approaches because they deemed them *easier* to use by end users [3, 4, 5, 6, 7, 41, 43, 46, 49, 53, 58, 61, 65, 66, 72, 73]. For instance, Zhang et al. [65] propose a tool that allows developers to draw (e.g. via tablets, digital pens) simple sketches on app screenshots. The tool then uses CV algorithms to analyze the shapes and structure of these hand drawn sketches to decode the meaning of each sketch. The tool then uses the sketch as a visual test spec to automatically generate a number of GUI test suites for mobile applications. The authors argue, and demonstrate, that providing developers with the option of using simple hand sketches to automatically generate test cases is a more natural and easier to use approach to create test cases.

This viewpoint is also evident in papers targeting the detection of cross-browser incompatibilities (XBIs) [4, 5, 6, 41, 46, 61, 72]. This problem requires developers to detect visual differences between web pages within the same web application when rendered on different browsers. This task is challenging for a number of reasons. First, manually performing this task, for instance through eyeballing, is neither efficient nor easy. Hence, an automatic technique would require simulating the reasoning that humans do while *seeing and comparing* two web pages. This can be easily simulated through a computer vision technique called *image differencing*, for which a plethora of different techniques have been proposed.

However, originally, the same problem was tackled from a non-visual perspective. First works on XBIs used well-known DOM differencing techniques as a proxy for finding visual defects. The main limitation was the fact that DOM-level differences do not always correspond to a different visual layout. Hence, this caused such techniques to have many false positives and a low accuracy. On the other hand, visual approaches have been proposed both as an alterna-

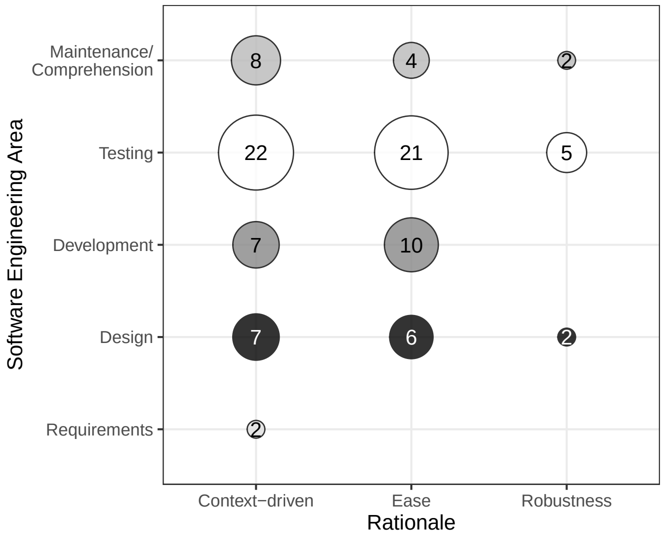


Fig. 7: Distribution of rationales per SE area.

tive, as well as, a complementary technique to overcome the limitations of the DOM-based approaches.

4.3.3 Robustness

The concept of robustness of a visual approach concerns its capability of maintaining its effectiveness despite minor visual changes happening in the software being analyzed.

A few papers [1, 2, 80] mention robustness as rationale for choosing computer vision. They explain that visual approaches are used because they are considered more change-tolerant than alternative code-based techniques. In other words, according to the authors, using a code-based approach for the same problem is likely to produce a fragile tool that would require a high maintenance cost.

For instance, Chang et al. [1] describe how spatial rearrangements of GUI components on the page can lead to fragile test scripts, a well-known issue in web testing [92]. According to Chang et al. [1], visual approaches are more robust to minor layout changes and elements repositioning. This viewpoint is discussed and confirmed in the paper by Alégroth et al. [2], in which image recognition of GUI elements allows the development of more robust system-level automated test suites. A similar finding is in the paper by Stocco et al. [8], in which an image processing pipeline was used to automatically trace web elements across different versions of the same web applications. Their tool VISTA exhibited a high test repair rate during software evolution, outperforming a DOM-based test repair solution. This essentially means that in the web domain, web app GUIs exhibit less frequent changes as compared to the DOM, as acknowledged by other researchers [93, 94, 95, 96].

The bubble chart of Figure 7 shows the distribution of the rationales in relation to the SE areas presented in Section 4.2. We notice that the context-driven category dominates across all SE areas. In the areas of requirements, design, and maintenance, visual approaches were prevalently used because, at this stage of the software development lifecycle, designers or requirements engineers mostly deal with visual abstractions of (portions) the software such as GUI mockups, or UML models. Computer vision allows the transformation of these visual artifacts to support successive SE tasks. For example, in the work by Zhang et al. [65], annotated sketches are used to specify test requirements and test case creation. In this case, the targeted SE area is both requirements engineering and testing.

4.3.4 Summary

In this section, the goal was to understand the rationales and motivations for using computer vision to address software engineering problems. An understanding of the rationales can help researchers decide if their research area or topic has similar problems or challenges, and then potentially explore using computer vision for their problem. We identified three categories of rationales: context-driven, ease of use, and robustness. Papers were classified as context-driven when the context of the software engineering problem itself has dictated the use of visual approaches. The ease of use classification was used in cases where the motivation is not necessarily driven by the context, but driven by the motivation of making an existing software engineering process easier to use (e.g. has less manual work, easier to comprehend). Finally, papers were classified as motivated by robustness whenever the motivation is making a software engineering task more accurate or less fragile. Out of all three categories of rationales, the context-driven category was the most common.

4.4 Computer Vision Techniques (RQ3)

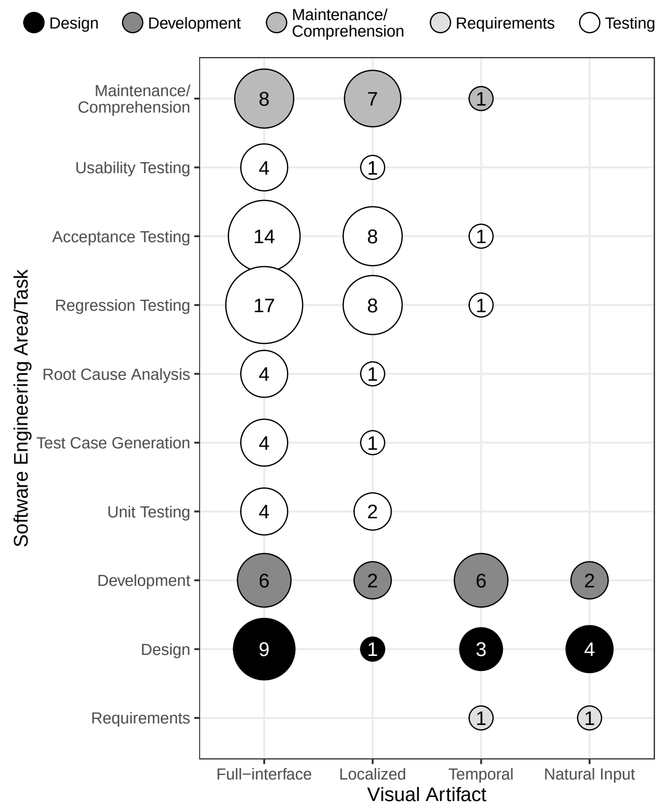
In order to investigate how computer vision is applied, we studied: (1) what visual artifacts are used, generated, or extracted from the software, and (2) what computer vision techniques are used to process or analyze the visual artifacts. We recall that visual artifacts are visual data (e.g., images) used by one or more computer vision techniques, with the final objectives of addressing a software engineering problem.

4.4.1 Artifact Categories

Through our survey of the field, we classified the visual artifacts used in the literature into four categories: (1) full-interface artifacts, (2) localized artifacts, (3) temporal artifacts, and (4) natural input artifacts. Figure 8 shows the distribution of such visual artifacts with respect to the SE area/tasks.

The first category, *full-interface visual artifacts*, typically represents screenshots of the entire user interface of the system (whether a web browser or desktop application, as well as other forms such as visual content of TVs and car displays) [4, 5, 41, 43, 46, 51, 53, 54, 56, 58, 59, 60, 61, 62, 63, 65, 66, 69, 72, 73]. This type of artifact simply has one large screenshot that captures the entire interface. This artifact has been used most commonly to capture the *visual state* of the application (regardless of the platform), and analyzed further to perform various forms of testing (e.g., regression, acceptance, or test generation).

However, full-interface artifacts capture the visual state of the system at a coarse-grained level of granularity, which renders them less applicable when a more detailed analysis is needed. For these cases, our survey revealed



Software Engineering Area/Task	Full−interface	Localized	Temporal	Natural Input
Maintenance/ Comprehension	8	7	1	
Usability Testing	4	1		
Acceptance Testing	14	8	1	
Regression Testing	17	8	1	
Root Cause Analysis	4	1		
Test Case Generation	4	1		
Unit Testing	4	2		
Development	6	2	6	2
Design	9	1	3	4
Requirements			1	1

Fig. 8: Distribution of visual artifacts per SE area & task.

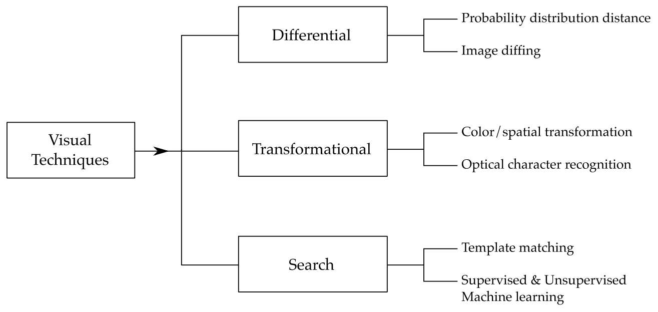

Visual Techniques
Differential
Probability distribution distance
Image diffing
Transformational
Color/spatial transformation
Optical character recognition
Search
Template matching
Supervised & Unsupervised Machine learning

Fig. 9: Synthesized taxonomy of visual techniques.

the more fine-grained category of *localized visual artifacts*. In this case, visual artifacts are created at the level of a specific component, an area of interest, or a certain feature [1, 2, 6, 7, 49, 52, 70, 71]. Compared to full-interface visual artifacts, this type of artifact is more beneficial for scenarios where analysis needs to be performed for a specific component or feature in a system. For instance, localized visual artifacts have been used to create a test case for a GUI (by recording and tracking a single visual artifact for UI elements) [1], or in debugging the rendering or capturing the behaviour of a specific HTML element in a web application [52].

The third category that emerged is *temporal artifacts*, in which the visual information captures the *dynamic behaviour* of some sequence or chain of information, states, or events [3, 9, 57, 64]. For instance, Bao et al. [64] use a temporal artifact (a video screen recording) to construct a tool to help researchers conducting user studies of developers' behaviours to automatically distill and transcribe their actions, inputs, and event sequences by capturing a video screen recording of their work session.

Finally, the last identified category is *natural input artifacts*. These artifacts capture a natural representation or interaction with a human user. The only example of this artifact that we found in the collected literature are hand-sketches [10, 68]. This type of artifact provides a number of benefits: (1) it provides a more intuitive and natural way for software engineers to interact with, design, develop, or test their software, and (2) it allows a broad degree of freedom in capturing user input, which can be useful when modelling or analyzing multi-variable complex systems.

4.4.2 Visual Techniques Taxonomy

In this section, we describe a taxonomy of visual techniques. The taxonomy was synthesized from the pool of papers we collected in this survey. This taxonomy has not been used elsewhere, since there are no existing surveys on the use of computer vision techniques in software engineering. For each paper, we analyzed the text and noted down the computer vision algorithms utilized by the authors to solve the software engineering problem being tackled. After conducting this process over all collected papers, we grouped the algorithms and identified three main patterns of algorithms, which are then collected in a taxonomy. We built the taxonomy in an iterative manner, where a new taxonomy category is defined if a certain computer vision algorithm can not be classified under any of the existing categories.

Figure 9 shows the resulting taxonomy. As shown in the figure, we identified three main categories of visual techniques used in software engineering research: (1) differential, (2) transformational, and (3) search techniques, which we will describe shortly. Each technique has sub-categories of algorithms, which will be described in Section 4.4.3.

Differential techniques are utilized to have two or more visual artifacts contrasted and their differences extracted [2, 3, 5, 6, 7, 41, 43, 46, 49, 52, 53, 54, 58, 59, 61, 62, 69, 72]. The nature of these differences is typically case-specific and often targets a specific feature that the paper is considering. Differential techniques are commonly used in situations where a comparison of different aspects, options, or versions of the software is desired. For instance, they have been widely utilized for cross-browser testing [6, 41, 46, 61, 72], where the goal is to check for any differences between two or more browsers in the way they render a web app.

The second category, *transformational techniques*, achieve a specific software engineering task by transforming the visual artifact into a more abstract type of information [4, 10, 51, 56, 57, 60, 63, 64, 65, 68, 71, 73]. This transformation is typically case-specific, as the higher-level abstract information is used to solve the specific instance of problems addressed in the paper. For instance, this approach has been used to allow manual hand-drawn strokes as a method to specify test executions and requirements [10], where transformational techniques are applied on hand-drawn stroke

TABLE 5: Major computer vision algorithms used in the collected papers.

Visual Technique	Algorithm	Description	Utilized in
Differential	Image diff	A processing method whose output is a function of the difference between a pair of input images (e.g. PID–Perceptual Image Differencing [97], PHash–Perceptual Hashing [98]).	[6, 7, 9, 41, 43, 48, 50, 52, 53, 56, 57, 58, 62, 64, 66, 69, 72, 73, 75, 78, 82, 87]
	Probability distribution distance	Measuring the distance between *distributions* of pixels in a pair of images (e.g. χ^2 distribution distance).	[3, 5, 41, 46, 54, 59, 61, 72, 81, 82]
Transformational	Color/Spatial transformation	Applying a transformation matrix to the spatial or color-space of one or more images, or transforming spatial regions into abstract data.	[9, 37, 38, 39, 40, 42, 44, 45, 47, 50, 55, 60, 63, 68, 69, 71, 75, 79, 83, 84, 85, 86]
	Optical character recognition	Recognizing images of strings and converting them into textual data.	[1, 49, 51, 57, 74, 80, 85, 88]
Search	Template matching	Finding where an image is located within another image.	[1, 2, 3, 4, 8, 42, 47, 64, 66, 70, 74, 80, 88]
	Machine learning	Finding closest visual matches or categories based on learning patterns in visual data.	[10, 36, 38, 56, 65, 67, 68, 76, 77, 87, 89, 90, 91]

instances to extract testing instructions and assertions from the strokes, and finally generate a working test case.

Finally, the third category is *search techniques* [1, 9, 66, 70]. In this case, a visual artifact is used as a key to find information within a larger set of visual artifacts. A popular example of this approach is visual record-and-playback tools such as Sikuli [1]. These tools first record component visual artifacts for every GUI element clicked or interacted with by a developer or user, and then, in the playback phase, a visual search method is employed to locate the element on screen to perform the recorded action (e.g., click).

In order to have a better insight on the use of different categories of visual techniques, the bubble plot of Figure 10 shows their distribution over the various SE areas. In the plot, software testing has a finer-grained granularity since it is by far the most represented area in our final pool of papers. We make a number of observations from the plot. First, we notice that transformational techniques are relatively the most ubiquitous, as they are quite uniformly represented across all SE areas and tasks. This is an expected result since transformational techniques provide means for extracting task-specific data from the visual artifacts, which makes them conveniently applicable to a large variety of problems. Next, we also notice that differential techniques were more specifically instrumental to testing tasks. Regression and acceptance testing greatly benefit from differential techniques, since they are very well aligned towards detecting differences and therefore suitable for indicating regression faults.

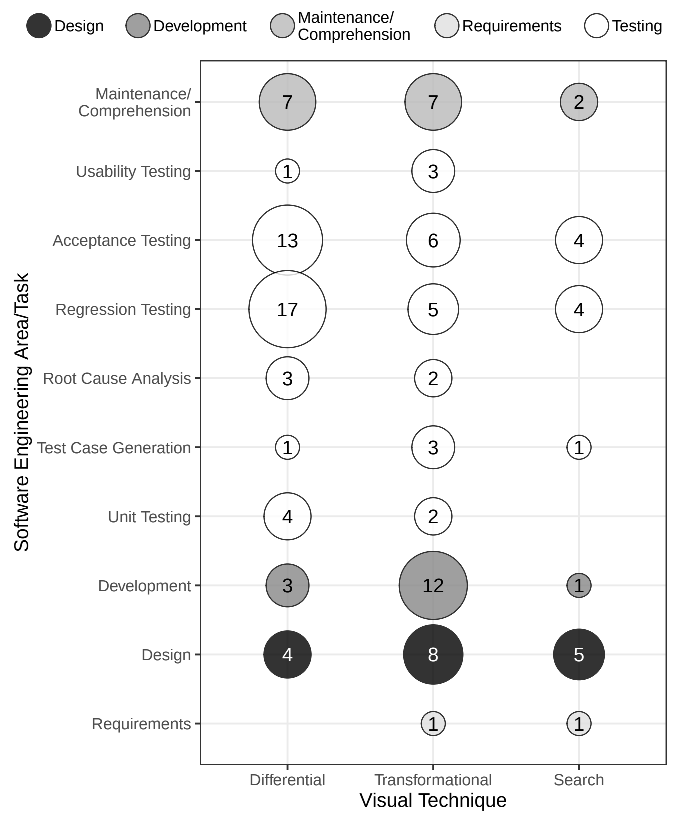


Fig. 10: Distribution of visual techniques per SE area & task.

4.4.3 Computer Vision Algorithms

In addition to the preceding analysis of the high-level categories of visual techniques, we also examine the specific computer vision algorithms used in the collected pool of papers. Table 5 shows the algorithms that have emerged from our analysis. We now describe each of the identified algorithms.

Image Differencing. In these CV algorithms, a pair of images is taken as input and the output is defined as a function of the difference between the pair. Various instances of these algorithms differ in their choice of the output function. For instance, the most common variation uses the raw absolute difference as the output value [7, 9, 41, 57, 62, 64, 66, 69, 75], where it is useful for faithfully detecting any *pixel-level* difference. Another variation adopts a more relaxed approach where the output function captures only *perceivable* differences by humans using algorithms such as PID (Perceptual Image Differencing) [97], PHash (Perceptual Hashing) [98], and Structural Similarity Index (SSIM) [99], which aim to reduce false positives by mimicking human perception. These techniques were utilized in [52, 53, 58, 72, 75, 82].

Probability Distribution Distance. These algorithms quantify distances in *populations* of pixels, as opposed to a pixel-wise comparison. The goal here is to measure how similar two given distributions are, such as image *histograms* [100] which give the distribution of pixels in an image. This is then used to establish whether the two distributions of pixels can be assumed to represent similar visual information. The χ^2 histogram distance (for both coloured and grayscale data) is by far the most commonly used distribution distance in our pool of papers [5, 46, 54, 59, 61, 82], as it is readily available in many implementations and provides a simple and effective approach for the needed quantification of distance.

Color/Spatial Transformation. These algorithms perform a transformation of the spatial or color-space of one or more images [9, 60, 63, 69, 71, 75, 79, 85]. This constitutes applying a 2D or 3D transformation matrix on the desired geometric space (e.g., a rearrangement of color-space). This class of algorithms has been used in our pool of papers to perform tasks such as extracting structure from images, aligning images, and various forms of thresholding to extract content.

Template Matching. Another major class of CV algorithms used in the papers is template matching. Here, one image is searched within another image or set of images. That is, a visual scan is performed to find a template image (hence the name) in a larger image or set of images. Several papers in our pool [1, 3, 4, 8, 64, 66, 70, 80] have used this approach to achieve tasks such as locating and finding coordinates of components and checking the presence/absence of components or certain features within a set of GUIs.

Optical Character Recognition (OCR). OCR algorithms use a series of computer vision analyses to recognize strings in images. Once the string is recognized, another sequence of steps converts each character in the image to textual data. We found multiple papers in our pool (e.g., [1, 49, 51, 57, 74, 80, 85]) which have used OCR for tasks such as checking GUI component labels and generating component labels from mockups/screenshots.

Machine Learning. Machine learning has also been used by a few papers in our pool. For instance, *decision trees* were used for classifying web pages in cross-browser testing [4, 46]. *Convolutional neural networks* (CNNs) were also used to analyze GUIs and their content [4, 79]. Furthermore, scale- and transformation-invariant features (e.g., SURF–Speeded-Up Robust Features [101], Wavelets [102]), which are basically features aiming at analyzing image structure and content, were used to classify and detect GUI elements [8, 64, 73, 85].

4.4.4 Libraries and Tools

We also investigated what industrial or open-source libraries and tools were used by the papers in their implementation of computer vision techniques. Table 6 shows a list of the CV tools or libraries that have been used by the papers in our pool. The first column reports the name of the tool or library. The second column describes the scope of the library, and the last column shows the paper(s) that utilize a certain library to implement their CV analysis or processing. Papers that do not state which library or tool they used are not included in the table, and therefore the number of papers in the table is smaller than the total pool. For the sake of completeness, we also included other CV tools and libraries that have not been used by any of the papers in our pool, which are marked by the "N/A" (not applicable) value in the last column. Due to the absence of queryable databases in which computer vision tools are listed, we resorted to manual search engine queries to find computer vision tools or libraries other than those already used by our pool. The queries we used were of the form "alternative (libraries or tools) to X", where X is one of the libraries already in our pool.

TABLE 6: Open-source and industrial computer vision tools and libraries utilized by papers in the pool.

Name	**Scope**	**Utilized in**
OpenCV	provides data structures and algorithms for a wide variety of advanced computer vision processing	[1, 3, 5, 8, 41, 46, 47, 50, 51, 53, 58, 59, 64, 70, 79, 80, 85, 87]
Tesseract	extensible and modular open-source optical character recognition (OCR) engine	[51, 57, 79, 88]
FineReader	comprehensive OCR engine that includes relevant pre/post-processing steps and formats	[74]
BoofCV	performance-oriented library that focuses on real-time processing	[57]
ImageMagick	simple and easy to use tool and library for basic image transformations and analysis of color	[7]
ITK (Insight ToolKit)	specialized in image and coordinates matching, with comprehensive support for high-dimensional data	N/A
VLFeat	geared towards feature extraction, covering a large set of feature descriptors	N/A
ImageJ	a modular tool and library with an extensive number of plugins for various image analysis tasks	N/A
Amazon Rekognition, Google Vision AI, Azure CV	cloud-based tools with a large number of pretrained analysis and recognition models	N/A

The most commonly used library is *OpenCV*. [1] This library has implementations for a large number of CV algorithms, including all the algorithms we report in this survey (Section 4.4.3). It was first released in 2000, and is still in active development, with the latest release (as of this writing) in July 2020.

Other than OpenCV, a few other libraries were used by only a handful of papers. *ImageMagick*[2] is a simple and easy to use library offering basic quick image manipulations (e.g., resize, rotate). *ImageJ*[3] is also quite similar in features

1. https://opencv.org
2. https://imagemagick.org/
3. https://imagej.net

and scope. *Tesseract*[4] is a popular open-source library that is modular and extensible, with support for more than 100 languages. *Abby FineReader*[5] is a commercial library that focuses specifically on OCR and includes many related preprocessing/postprocessing algorithms, as well as various file formats. *BoofCV*[6] focuses on performance-tuned algorithms and is geared towards cases where real-time response is priority. *ITK*[7] focuses on high-dimensional visual data often present statistics and science, and also has extensive image coordinates matching algorithms that might be beneficial in a number of image matching applications. *VLFeat*[8] is geared towards implementing a wide variety of feature extraction and matching algorithms, enabling applications in image search or transformation. Finally, there are cloud-based tools offered by major cloud hosting providers (e.g. Amazon Web Services, Google Cloud Platform, Microsoft Azure). The defining feature of these services is their cloud nature and the availability of many pre-trained computer vision models for various visual tasks, such as content tagging and visual path analysis.

4.4.5 Summary

The goal of this section was to explore what computer vision techniques were used and what visual artifacts were extracted from the software. To this end, we identified four categories of artifacts. The first category, full-interface artifacts, represents cases where the entire visual content of a software is used (e.g., entire desktop interface, entire console). The second category is localized artifacts, where only a specific module or component of the software is captured visually. Third, temporal artifacts capture the dynamic behavior of the states or events in a software. Finally, natural input artifacts capture natural forms of input by humans (e.g., hand sketches). Visual artifacts are then processed by one or more of three visual techniques. The first category, differential techniques, are based on various forms of contrasting two or more visual artifacts and using the differences to solve a software engineering problem. The second category, transformational techniques, rely on transforming the visual artifact into a more abstract data structure on which further analysis can be conducted. Finally, in search techniques, a visual artifact is used as a key to find information within a larger set of visual artifacts.

Table 5 shows the major computer vision algorithms used in the collected papers. The visual technique column describes the high-level goal of what the algorithm is trying to achieve visually. The algorithm column lists the specific algorithms that were used to achieve the task. For instance, when papers wanted to do a differential examination of visual artifacts, the two computer vision approaches that were used were image diffing and distribution distances.

4.5 Evaluation and Challenges (RQ4)

4.5.1 Evaluation Methods

We systematically studied the selected papers to understand the most common methods used for evaluating the proposed visual approaches. Table 7 (Evaluation Methods) depicts the results: the evaluation methods presented are not necessarily disjoint, since one technique can for instance be evaluated with respect to its performance (i.e., running time), and its accuracy calculated by the judgment of human participants.

In more than half of the works (53%), the evaluation methods were performed using wide-spread effectiveness assessment measures such as precision and recall [3, 5, 41, 46, 51, 53, 54, 59, 60, 61, 63, 64, 66, 68, 69, 71, 72, 73].

Another significant body of work measured the manual effort saved by the visual approach with respect to the humanly performed task [4, 9, 9, 43, 53, 59, 66, 68, 70]. Moreover, several works also evaluated the performance of the proposed approaches, usually measuring the running time on common developer platforms (i.e., mid-level notebooks) [6, 51, 54, 60, 63, 64, 68, 69].

Performance is potentially considered as important as the accuracy because image analysis techniques are deemed as being computationally expensive. Thus, their application and use must carefully evaluate the overhead imposed by these visual approaches over the most general SE tool being developed. Initially, this might not favor their adoption if compared to other less computationally-intensive approaches. However, the authors report that the time taken by their proposed techniques are all reasonable in common processing environments, suggesting that execution time should not be considered as a deterrent decision criterion when adopting a visual approach to support a SE task.

In several works, the human expertise was used to assess the output of the visual techniques [5, 41, 46, 53, 54, 57, 58, 72, 73]. As an example, Mahajan and Halfond [53] examine the efficacy of WebSee (a tool that identifies presentational failures in HTML pages) through manual investigation of its output when it is executed on 253 automatically-generated test cases. This is done to see whether WebSee is able to correctly identify the faulty HTML elements seeded when generating the test cases. While we were expecting more human judgment involved in the evaluation of the selected papers, its application depends largely on the context and the problem being solved.

Four works do not propose a quantitative empirical evaluation of the proposed technique, but rather present them by means of use cases. A closer look at these publications revealed that two of them are short papers [2, 10], where having a more thorough evaluation is not mandatory. The other two papers [52, 56] were published at the ACM Symposium on User Interface Software and Technology (UIST), in which use case demonstrations appear to be more frequent. Four works included an industrial evaluation [9, 49, 53, 73] where the work is evaluated through feedback or measurements in an industrial setting.

Finally, we noticed diversity in the evaluation methods of techniques aimed at solving similar problems. For example, several works have been proposed to identify some sort of *visual defects* (e.g., all the approaches dealing with XBI). Four of these techniques [7, 53, 58, 60] use "seeded faults" as an approach for constructing test oracles, whereas others take a different evaluation path. For instance, Roy Choudhary et al. [5] manually count all XBIs detected by different tools (i.e., their agreement) as an upper bound for

4. https://github.com/tesseract-ocr/tesseract
5. https://abbyy.com/
6. http://boofcv.org
7. https://itk.org/
8. https://vlfeat.org/

TABLE 7: Evaluation methods and challenges.

Evaluation Methods	
Accuracy Measures	[3, 5, 8, 38, 41, 46, 51, 53, 54, 59, 60, 61, 63, 64, 66, 68, 69, 71, 72, 73, 75, 76, 77, 79, 80, 81, 82, 83, 84, 85, 86, 87, 89, 91]
Comparison against manual work	[4, 9, 9, 40, 43, 48, 53, 59, 66, 68, 70, 75, 79, 80, 81, 82, 90]
Survey or manual validation of results	[5, 37, 41, 46, 48, 53, 54, 57, 58, 67, 72, 73, 74, 76, 78, 86, 88, 91]
Time comparison	[6, 8, 37, 38, 51, 54, 60, 63, 64, 68, 69, 74, 76, 80, 83]
Comparison against competitor approaches	[2, 3, 8, 37, 39, 58, 61, 62, 65, 72]
Comparison to original design/output	[51, 60, 63, 70, 71]
Use case demonstration	[2, 10, 47, 50, 52, 56]
Industry context/feedback	[9, 39, 49, 53, 73, 74, 86]
Comparison with random/brute force	[6, 7, 59, 60]
Seeded faults	[7, 53, 58, 60]
Differential comparisons	[1, 58]
Line/state coverage	[62, 65]
Comparison on different physical devices	[3, 55]
Challenges	
Noise	[3, 8, 9, 37, 38, 39, 50, 51, 55, 57, 64, 68, 69, 70, 73, 75, 79, 80, 82, 83, 86]
Dynamicity	[1, 40, 41, 48, 60, 63, 71, 74, 78]
Recognition	[8, 10, 37, 38, 39, 47, 50, 64, 65, 67, 69, 73, 74, 75, 76, 77, 79, 80, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91]
Visibility	[1, 41, 64, 65, 70, 74, 78]

the number of issues that an XBI detection tool should identify, and use this number to compute the recall of X-PERT. As a further evaluation, authors could have been also evaluating their technique using seeded XBIs to broaden the evaluation. Conversely, the works that only use seeded faults could use the agreement of multiple fault detectors to compute recall. This scenario essentially suggests that the evaluation of some of the proposed techniques in the literature can be substantially improved by leveraging the evaluation approaches from other techniques that aim at solving similar problems.

4.5.2 Challenges and Limitations

Evaluations also exposed the main challenges and limitations that authors faced while developing their visual-based solutions. Unfortunately, a subset of the papers (41%) either do not explicitly discuss drawbacks for the visual approach being evaluated, or do not provide concrete failing examples. On the other hand, the majority of the papers (59%) do report the challenges encountered during the experimentations, summarized in Table 7 (Challenges), which we describe next.

Noise. The most recurring technical issue that inhibited the correct functioning of visual approaches concerns the *noise* present in the visual artifacts [3, 9, 51, 57, 64, 68, 70, 73]. Noise refers to the presence of other random or overlapping items in the background of the visual artifact that prevents, for instance, the correct detection of the target object. Hence, before applying the desired visual method, a pre-processing technique is often used to clear the noise out of the artifact.

As an example, Ponzanelli et al. [57] apply OCR to detect source code in frames sampled from video tutorials. However, the effectiveness of OCR fluctuates dramatically if the considered frame's background contains non-pertinent information. As a solution, the authors utilize two additional visual techniques—shape detection and frame segmentation—in order to focus OCR towards the area of interest.

Another more specific noise-related limitation pertains to the *sensitivity* of the visual approach to theme/surroundings changes, lighting conditions, and reflections [1, 3, 9, 69, 70, 73]. A possible mitigation concerns using threshold parameters to limit their detrimental effect.

Dynamicity. Highly-variable visual artifacts are also a major challenge for the design and development of reliable visual approaches.

One source of fragility is due to animations [1, 41, 71]. Indeed, a continuously-changing visual artifact (e.g., a video frame) represents a major limitation for many one-shot techniques. As a possible solution, such techniques would need to be applied repeatedly, similarly as real-time domains, or, in extreme cases, re-designed from scratch to meet the new domain requirements.

Another source of fragility is due to web advertisements [63]. An ad is rendered as a dynamic component within a more static container (i.e., a web page), the visual representation of which usually changes across consecutive loads of the web page or over different time stamps. Visual methods need to isolate the dynamics of web pages to avoid erroneous behaviour or false positives.

Recognition. Another challenge of visual approaches that emerged from our study concerns (1) accurately identifying small imperceptible differences between images and (2) recognizing complex user-defined actions such as manually inserted strokes or handwriting.

For example, a source of false positives in VISOR [69], an image comparison technique used for black-box testing of digital TVs, is the tiny differences that occur when rendering accent characters on the screen (e.g., cedillas). Typically, the choice on whether such differences should be considered as defects or not is left to human's perception. Two other works experienced recognition issues, but not at the same pixel-level as VISOR. Bao et al. [64] discuss problems in detecting visual fine-grained developers' actions from videos, such as code editing, text selection, and window scrolling. Similarly, Scharf and Amma [10] mention issues in detecting handwriting in manually-produced sketches.

These findings demonstrate how complex is the set of visual differences that can emerge while comparing two images, and how vast and multifaceted is the set of input actions that are possible on a user interface. Indeed, when a visual method is used to actively support the *end user*, they arguably need to recognize a broader set of inputs than those they would receive from another algorithm, if they were executed in a software controlled environment. This poses new challenges to the development of robust visual approaches for aiding SE.

Visibility. At last, in four works, having the visual artifact present in the main rendering area is mentioned as a requirement for the visual technique to effectively fulfill its task. This requirement can be violated by, e.g., *fading* [1, 70] and *scrolling* [41, 64].

As an example, Leotta et al. [70]'s tool PESTO uses template matching to detect the optimal visual locator corresponding to a web element on the page. However, the web element of interest might be outside the actual rendered web page. This happens when a long and complex form with multiple fields cannot be entirely visualized within the main screen. The authors propose an engineering workaround to solve issues related to scrolling: if the visual artifact being searched is not immediately displayed, the tool scrolls the page down automatically and the template matching is repeated.

5 Discussion

Increasing Adoption of Visual Approaches. Our findings show a general growth trend in the adoption and use of visual approaches in the SE community. Most of the surveyed papers explicitly recognize, and empirically demonstrate, the contribution brought by visual methods in supporting SE tasks.

Based on our examination of the literature, we attribute this increase to two factors. First, a sizeable number of software developed nowadays have a GUI or other visual interfaces. The end-user experience is increasingly becoming more important in adding value to software, and therefore the adoption of visual methods is expected to increase further in the next years. Our examination of the trend of number of annual publications already shows this trend of increasing number of works utilizing visual techniques. Second, the rapid pace of improvement in hardware and processor architecture has made the efficiency and run time of advanced visual techniques feasible in common development environments, which we expect would cause an increased adoption of visual approaches further down the line.

Software Testing a Major Driver of Visual Approaches. The majority of papers (around 75%) have focused on the research area of software testing. Our intuition behind this is two-fold. First, testing is one of the most active SE research areas in general, so it is not surprising that most of the collected papers fall within this category. Second, testing is largely a tooling-based research area, in which tool prototypes are developed and empirically evaluated. Many different static and dynamic analysis tools are proposed each year to facilitate test engineers' activities. The results of this survey show that the visual perspective of the software has been recently used to complement static and dynamic analysis because it provides a novel and complimentary perspective of the software under test. The types of analyses that are performed on the presentation-level of the application would be likely very difficult to perform by analyzing the source code only, especially with the increasingly complex interfaces and the great emphasis placed on user experience, of which the interface is a cornerstone component.

Custom Solutions. The visual techniques used in the collected papers are often ad-hoc solutions developed for tackling a specific problem. All collected papers have discussed, to some extent, the need of visual approaches for parameter fine-tuning, such as optimal threshold selections. Authors recognize that this requirement is unlikely to be solved by a consolidated and broadly-accepted solution. In fact, manipulating visual artifacts through a visual technique is highly application-specific, both in the adopted approach and in the considered domain [103].

For instance, this survey highlights a large body of work in the area of cross-browser incompatibility. The authors of these papers have adopted a large variety of solutions (or incremental variations) to tackle the same problem. To mention a few, Choudhary et al. [41] use an image comparison measure based on Earth Mover's Distance (EMD), whereas Mahajan and Halfond [53] adopt perceptual differencing, and He et al. [61] compare the colour histograms, among other approaches. This trend can be partially explained by the need of proposing and experimenting with novel and potentially useful techniques. However, a researcher approaching this topic for the first time could be somewhat disoriented. In fact, given that the solutions for the same problem are many, and they are often evaluated on different benchmarks, it is not straightforward to find an agreement on what the best technique could be. This led to a landscape where each work would typically experiment with a custom visual processing pipeline to address the specifics of the SE task at hand.

Need for Visual Benchmarks in SE. We highlight the lack of comparative visual benchmarks on which to evaluate the plethora of visual approaches utilized in software engineering research. A repository of standard, well-organized, categorized, and labeled visual artifacts could be very useful to support empirical experiments, and to guide the next generation of research utilizing visual approaches for software engineering tasks. Such repositories exist in traditional (non-visual) software engineering research, such as SIR [104], Defects4J [105], SF100 [106], and BugsJS [107, 108]. This has not been the case, however, for visual techniques in software engineering. For instance, having analogous repositories for visual bugs can foster further applications of visual methods in software testing.

Similarly, object detection and classification tasks need labeled images. In computer vision literature, there exist some pre-validated and labeled visual benchmarks, such as ImageNet [109], BSDS500 [110], or Caltech 101 [111]. In software engineering, a benchmark of labeled visual artifacts might aid in developing visual techniques, or training systems for machine learning and deep learning scenarios. A notable step in this direction has been carried out in the Rico [62] repository. The repository contains around 70k labeled UI screenshots, each of which are labeled with visual, textual, structural, and interaction trace data. The dataset facilitates software engineering tasks related to the UI, such as UI design search, UI layout generation, and UI code generation.

Maintainability of Visual Artifacts. The visual artifacts created or extracted from the software are rarely static across time, especially for rapidly evolving software such as in agile environments. The artifacts would therefore have to be frequently modified or updated to keep track of the underlying evolving software. Alégroth et al. [2] indicate that the maintainability of visual artifacts produced and used by the visual testing tools as being a major challenge of the visual-based testing approaches. Potential research directions to mitigate this challenge include proposing strategies for conducting cost-benefit analysis depending on the expected degree of visual evolution of the software, and devising automated techniques to help with or reduce the maintainability effort for visual artifacts.

Familiarity with Computer Vision. Perhaps the biggest challenge hindering a wider adoption of visual approaches in software engineering is the lack of familiarity with computer vision techniques. For instance, Delamaro et al. [43] describe how developers should have basic knowledge of image processing in order to even *use* the proposed tool in the paper. This is because the visual artifacts can structurally vary with each use, and thus sometimes one or more manual image processing adjustments need to be performed before being able to process the visual artifacts.

5.1 Threats to Validity

The main threats to validity of this survey are the bias in the papers' selection and misclassification. We mitigate these threats as follows.

Our paper selection was driven by the keywords related to visual approaches and software engineering (see Section 3.4). We may have missed studies that use visual methods in the software engineering activities that are not captured by our terms list. To mitigate this threat, we performed an issue-by-issue manual search of the major software engineering conferences and journals, and followed through with a snowballing process.

Concerning the papers' classification, we manually classified all selected papers into different categories based on the targeted SE area, as well as, more fine-grained subcategories based on their domains, tasks, and the utilized visual methods (see Section 4). Identifying the rationale from the papers that do not explicitly mentioned it involved some subjectivity and may have resulted in suboptimal mappings, which constitutes another threat. However, there is no ground-truth labeling for such classification. To minimize classification errors, the first three authors carefully analyzed the full text and performed the classifications individually. Any disagreements were resolved by further discussion.

6 Conclusions

A recent and growing trend in software engineering research is to adopt a *visual perspective* of the software, which entails extracting and processing *visual artifacts* relevant to software using computer vision techniques. To gain a better understanding of this trend, in this paper, we surveyed the literature on the use of computer vision approaches in software engineering. From more than 2,716 publications, we systematically obtained 66 papers and analyzed them according to a number of research dimensions. Our study revealed that computer vision techniques have been utilized in all areas of software engineering, albeit more prevalently in the software testing field. We also discussed why computer vision is utilized, how these techniques are evaluated, and what limitations they bear. Our suggestions for future work include the development of common frameworks and visual benchmarks to collect and evaluate the state-of-the-art techniques, to avoid relying on ad-hoc solutions. We believe that the findings of this work illustrate the potential of visual approaches in software engineering, and may help newcomers to the field in better understanding the research landscape.

References

[1] T.-H. Chang, T. Yeh, and R. C. Miller, "GUI Testing Using Computer Vision," in *Proc. of CHI '10*, 2010, pp. 1535–1544.

[2] E. Alégroth, M. Nass, and H. H. Olsson, "JAutomate: A Tool for System- and Acceptance-test Automation," in *Proc. of ICST '13*, 2013, pp. 439–446.

[3] Y. D. Lin, J. F. Rojas, E. T. H. Chu, and Y. C. Lai, "On the Accuracy, Efficiency, and Reusability of Automated Test Oracles for Android Devices," *TSE*, vol. 40, no. 10, pp. 957–970, 2014.

[4] N. Semenenko, M. Dumas, and T. Saar, "Browserbite: Accurate Cross-Browser Testing via Machine Learning over Image Features," in *Proc. of ICSM '13*, 2013, pp. 528–531.

[5] S. Roy Choudhary, M. R. Prasad, and A. Orso, "X-PERT: Accurate Identification of Cross-browser Issues in Web Applications," in *Proc. of ICSE '13*, 2013, pp. 702–711.

[6] E. Selay, Z. Q. Zhou, and J. Zou, "Adaptive Random Testing for Image Comparison in Regression Web Testing," in *Proc. of DICTA '14*, ser. DICTA '14, 2014, pp. 1–7.

[7] S. Mahajan and W. G. Halfond, "Finding HTML Presentation Failures Using Image Comparison Techniques," in *Proc. of ASE '14*, 2014, pp. 91–96.

[8] A. Stocco, R. Yandrapally, and A. Mesbah, "Visual web test repair," in *Proceedings of the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2018, p. 12 pages.

[9] Y. Li, X. Cao, K. Everitt, M. Dixon, and J. A. Landay, "FrameWire: A Tool for Automatically Extracting Interaction Logic from Paper Prototyping Tests," in *Proc. of CHI '10*, 2010, pp. 503–512.

[10] A. Scharf and T. Amma, "Dynamic Injection of Sketching Features into GEF Based Diagram Editors," in *Proc. of ICSE '13*, 2013, pp. 822–831.

[11] A. Kumar, "Computer-vision-based fabric defect detection: A survey," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 1, pp. 348–363, 2008.

[12] C. Kanellakis and G. Nikolakopoulos, "Survey on computer vision for uavs: Current developments and trends," *Journal of Intelligent & Robotic Systems*, vol. 87, pp. 141–168, 2017.

[13] Y. Liu and Q. Dai, "A survey of computer vision applied in aerial robotic vehicles," in *2010 International Conference on Optics, Photonics and Energy Engineering (OPEE)*, vol. 1, 2010, pp. 277–280.

[14] A. Al-Kaff, D. Martín, F. García, A. de la Escalera, and J. María Armingol, "Survey of computer vision algorithms and applications for unmanned aerial vehicles," *Expert Systems with Applications*, vol. 92, pp. 447–463, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417417306395

[15] T. Gandhi and M. M. Trivedi, "Pedestrian collision avoidance systems: a survey of computer vision based recent studies," in *2006 IEEE Intelligent Transportation Systems Conference*, 2006, pp. 976–981.

[16] A. Brunetti, D. Buongiorno, G. F. Trotta, and V. Bevilacqua, "Computer vision and deep learning techniques for pedestrian detection and tracking: A survey," *Neurocomputing*, vol. 300, pp. 17–33, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092523121830290X

[17] J. Janai, F. Güney, A. Behl, and A. Geiger, "Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art," *CoRR*, vol. abs/1704.05519, 2017. [Online]. Available: http://arxiv.org/abs/1704.05519

[18] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, pp. 1–1, 2020.

[19] Á. Beszédes, "Interdisciplinary survey of fault localization techniques to aid software engineering," *Acta Polytechnica Hungarica*, vol. 16, pp. 207–226, 2019.

[20] D. van der Linden and I. Hadar, "A systematic literature review of applications of the physics of notations," *IEEE Transactions on Software Engineering*, vol. 45, no. 8, pp. 736–759, 2019.

[21] L. N. Sabaren, M. A. Mascheroni, C. L. Greiner, and E. Irrazábal, "A systematic literature review in cross-browser testing," *JCST*, vol. 18, no. 01, Apr. 2018.

[22] M. Wagner, F. Fischer, R. Luh, A. Haberson, A. Rind, D. A. Keim, W. Aigner, R. Borgo, F. Ganovelli, and I. Viola, "A survey of visualization systems for malware analysis." in *EuroVis (STARs)*, 2015, pp. 105–125.

[23] Y. Zhang, Y. Xiao, M. Chen, J. Zhang, and H. Deng, "A survey of security visualization for computer network logs," *Security and Communication Networks*, vol. 5, no. 4, pp. 404–421, 2012.

[24] P. Caserta and O. Zendra, "Visualization of the static aspects of software: A survey," *IEEE transactions on visualization and computer graphics*, vol. 17, no. 7, pp. 913–933, 2010.

[25] M.-A. D. Storey, D. Čubranić, and D. M. German, "On the use of visualization to support awareness of human activities in software development: a survey and a framework," in *Proceedings of the 2005 ACM symposium on Software visualization*. ACM, 2005, pp. 193–202.

[26] R. L. Novais, A. Torres, T. S. Mendes, M. Mendonça, and N. Zazworka, "Software evolution visualization: A systematic mapping study," *Information and Software Technology*, vol. 55, no. 11, pp. 1860–1883, 2013.

[27] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 15, no. 2, pp. 87–109, 2003.

[28] A. Issa, J. Sillito, and V. Garousi, "Visual testing of graphical user interfaces: An exploratory study towards systematic definitions and approaches," in *2012 14th IEEE International Symposium on Web Systems Evolution (WSE)*, Sep. 2012, pp. 11–15.

[29] E. Alégroth and R. Feldt, "On the long-term use of visual gui testing in industrial practice: a case study," *Empirical Software Engineering*, vol. 22, no. 6, pp. 2937–2971, 2017.

[30] E. Alégroth, Z. Gao, R. Oliveira, and A. Memon, "Conceptualization and evaluation of component-based testing unified with visual gui testing: an empirical study," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2015, pp. 1–10.

[31] E. Alégroth, A. Karlsson, and A. Radway, "Continuous integration and visual gui testing: Benefits and drawbacks in industrial practice," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, 2018, pp. 172–181.

[32] V. Garousi, W. Afzal, A. Çağlar, İ. B. Işık, B. Baydan, S. Çaylak, A. Z. Boyraz, B. Yolaçan, and K. Herkiloğlu, "Comparing automated visual gui testing tools: an industrial case study," in *Proceedings of the 8th ACM SIGSOFT International Workshop on Automated Software Testing*. ACM, 2017, pp. 21–28.

[33] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[34] I. C. Society, P. Bourque, and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, 3rd ed., 2014.

[35] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proc. of EASE '14*, 2014, pp. 1–10.

[36] J. A. Landay and B. A. Myers, "Sketching interfaces: Toward more human interface design," *Computer*, vol. 34, no. 3, pp. 56–64, 2001.

[37] A. Caetano, N. Goulart, M. Fonseca, and J. Jorge, "Javasketchit: Issues in sketching the look of user interfaces," in *AAAI Spring Symposium on Sketch Understanding*, 2002, pp. 9–14.

[38] J. Fails and D. Olsen, "A design tool for camera-based interaction," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 2003, pp. 449–456.

[39] A. Coyette, S. Kieffer, and J. Vanderdonckt, "Multi-fidelity prototyping of user interfaces," in *IFIP Conference on Human-Computer Interaction*. Springer, 2007, pp. 150–164.

[40] X. S. Zheng, I. Chakraborty, J. J.-W. Lin, and R. Rauschenberger, "Correlating low-level image statistics with users-rapid aesthetic and affective judgments of web pages," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 1–10.

[41] S. R. S. Choudhary, H. Versee, and A. Orso, "WEBD-

IFF: Automated Identification of Cross-browser Issues in Web Applications," in *Proc. of ICSM '10*, 2010, pp. 1–10.

[42] M. Dixon and J. Fogarty, "Prefab: implementing advanced behaviors using pixel-based reverse engineering of interface structure," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2010, pp. 1525–1534.

[43] M. E. Delamaro, L. dos Santos Nunes Fátima, and O. R. A. Paes, "Using concepts of content-based image retrieval to implement graphical testing oracles," *STVR*, vol. 23, no. 3, pp. 171–198, 2011.

[44] M. Dixon, D. Leventhal, and J. Fogarty, "Content and hierarchy in pixel-based methods for reverse engineering interface structure," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2011, pp. 969–978.

[45] J. Seifert, B. Pfleging, E. del Carmen Valderrama Bahamóndez, M. Hermes, E. Rukzio, and A. Schmidt, "Mobidev: A tool for creating apps on mobile phones," in *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*. ACM, 2011, pp. 109–112.

[46] S. R. Choudhary, M. R. Prasad, and A. Orso, "Cross-Check: Combining Crawling and Differencing to Better Detect Cross-browser Incompatibilities in Web Applications," in *Proc. of ICST '12*, 2012, pp. 171–180.

[47] P. Givens, A. Chakarov, S. Sankaranarayanan, and T. Yeh, "Exploring the internal state of user interfaces by combining computer vision techniques with grammatical inference," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 1165–1168.

[48] H.-S. Liang, K.-H. Kuo, P.-W. Lee, Y.-C. Chan, Y.-C. Lin, and M. Y. Chen, "Seess: seeing what i broke–visualizing change impact of cascading style sheets (css)," in *Proceedings of the 26th annual ACM symposium on User interface software and technology*, 2013, pp. 353–356.

[49] D. Amalfitano, A. R. Fasolino, S. Scala, and P. Tramontana, "Towards Automatic Model-in-the-loop Testing of Electronic Vehicle Information Centers," in *Proc. of WISE '14*, 2014, pp. 9–12.

[50] L. Bao, J. Li, Z. Xing, X. Wang, and B. Zhou, "scvripper: video scraping tool for modeling developers' behavior using interaction data," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 673–676.

[51] T. A. Nguyen and C. Csallner, "Reverse Engineering Mobile Application User Interfaces with REMAUI," in *Proc. of ASE '15*, 2015, pp. 248–259.

[52] B. Burg, A. J. Ko, and M. D. Ernst, "Explaining Visual Changes in Web Interfaces," in *Proc. of UIST '15*, 2015, pp. 259–268.

[53] S. Mahajan and W. G. J. Halfond, "Detection and Localization of HTML Presentation Failures Using Computer Vision-Based Techniques," in *Proc. of ICST '15*, 2015, pp. 1–10.

[54] A. Hori, S. Takada, H. Tanno, and M. Oinuma, "An Oracle based on Image Comparison for Regression Testing of Web Applications," in *Proc. of SEKE '15*, ser. SEKE '15, 2015, pp. 639–645.

[55] K. Reinecke, D. R. Flatla, and C. Brooks, "Enabling designers to foresee which colors users cannot see," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 2693–2704.

[56] B. Deka, Z. Huang, and R. Kumar, "ERICA: Interaction Mining Mobile Apps," in *Proc. of UIST '16*, 2016, pp. 767–776.

[57] L. Ponzanelli, G. Bavota, A. Mocci, M. D. Penta, R. Oliveto, M. Hasan, B. Russo, S. Haiduc, and M. Lanza, "Too Long; Didn't Watch! Extracting Relevant Fragments from Software Development Video Tutorials," in *Proc. of ICSE '16*, 2016, pp. 261–272.

[58] S. Mahajan, B. Li, P. Behnamghader, and W. G. J. Halfond, "Using Visual Symptoms for Debugging Presentation Failures in Web Applications," in *Proc. of ICST '16*, ser. ICST '16, 2016, pp. 191–201.

[59] Y. Feng, J. A. Jones, Z. Chen, and C. Fang, "Multi-objective Test Report Prioritization Using Image Understanding," in *Proc. of ASE '16*, 2016, pp. 202–213.

[60] M. Patrick, M. D. Castle, R. O. J. H. Stutt, and C. A. Gilligan, "Automatic Test Image Generation Using Procedural Noise," in *Proc. of ASE '16*, 2016, pp. 654–659.

[61] M. He, G. Wu, H. Tang, W. Chen, J. Wei, H. Zhong, and T. Huang, "X-Check: A Novel Cross-Browser Testing Service Based on Record/Replay," in *Proc. of ICWS '16*, 2016, pp. 123–130.

[62] B. Deka, Z. Huang, C. Franzen, J. Hibschman, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A Mobile App Dataset for Building Data-Driven Design Applications," in *Proc. of UIST '17*, 2017, pp. 845–854.

[63] M. Wan, Y. Jin, D. Jin, J. Gui, S. Mahajan, and W. G. J. Halfond, "Detecting display energy hotspots in android apps," vol. 27, no. 6, 2017.

[64] L. Bao, J. Li, Z. Xing, X. Wang, X. Xia, and B. Zhou, "Extracting and Analyzing Time-series HCI Data from Screen-captured Task Videos," *ESEM*, vol. 22, no. 1, pp. 134–174, 2017.

[65] C. Zhang, H. Cheng, E. Tang, X. Chen, L. Bu, and X. Li, "Sketch-guided GUI Test Generation for Mobile Applications," in *Proc. of ASE '17*, 2017, pp. 38–43.

[66] C.-F. R. Chen, M. Pistoia, C. Shi, P. Girolami, J. W. Ligman, and Y. Wang, "UI X-Ray: Interactive Mobile UI Testing Based on Computer Vision," in *Proc. of IUI '17*, 2017, pp. 245–255.

[67] S. Wu, J. Wieland, O. Farivar, and J. Schiller, "Automatic alt-text: Computer-generated image descriptions for blind users on a social network service," in *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, 2017, pp. 1180–1192.

[68] S. P. Reiss and Q. Miao, Yun Xin, "Seeking the user interface," *ASE*, vol. 25, no. 1, pp. 157–193, 2018.

[69] M. F. Kıraç, B. Aktemur, and H. Sözer, "VISOR: A fast image processing pipeline with scaling and translation invariance for test oracle automation of visual output systems," *JSS*, vol. 136, pp. 266–277, 2018.

[70] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "PESTO: Automated migration of DOM-based web tests towards the visual approach," *STVR*, vol. 28, no. 4, 2018.

[71] M. Bajammal and A. Mesbah, "Web canvas testing through visual inference," in *Proc. of ICST '18*, 2018.

[72] Z. Xu and J. Miller, "Cross-Browser Differences Detection Based on an Empirical Metric for Web Page Visual Similarity," *TOIT*, vol. 18, no. 3, pp. 1–23, 2018.

[73] T. Kuchta, T. Lutellier, E. Wong, L. Tan, and C. Cadar, "On the correctness of electronic documents: studying, finding, and localizing inconsistency bugs in pdf readers and files," *EMSE*, 2018.

[74] L. Bao, Z. Xing, X. Xia, and D. Lo, "Vt-revolution: Interactive programming video tutorial authoring and watching system," *IEEE Transactions on Software Engineering*, 2018.

[75] K. Moran, B. Li, C. Bernal-Cárdenas, D. Jelf, and D. Poshyvanyk, "Automated reporting of gui design violations for mobile apps," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 165–175.

[76] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 665–676.

[77] S.-H. Sun, H. Noh, S. Somasundaram, and J. Lim, "Neural program synthesis from diverse demonstration videos," in *International Conference on Machine Learning*, 2018, pp. 4790–4799.

[78] S. Lim, J. Hibschman, H. Zhang, and E. O'Rourke, "Ply: A visual web inspector for learning from professional webpages," in *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, 2018, pp. 991–1002.

[79] K. P. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk, "Machine learning-based prototyping of graphical user interfaces for mobile apps," *IEEE Transactions on Software Engineering*, 2018.

[80] H. Tanno and Y. Adachi, "Support for finding presentation failures by using computer vision techniques," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2018, pp. 356–363.

[81] M. Bajammal, D. Mazinanian, and A. Mesbah, "Generating reusable web components from mockups," in *Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering*, 2018.

[82] K. Moran, C. Watson, J. Hoskins, G. Purnell, and D. Poshyvanyk, "Detecting and summarizing gui changes in evolving mobile apps," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 543–553.

[83] S. Natarajan and C. Csallner, "P2a: A tool for converting pixels to animated mobile application user interfaces," in *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*. ACM, 2018, pp. 224–235.

[84] M. H. Osman, T. Ho-Quang, and M. Chaudron, "An automated approach for classifying reverse-engineered and forward-engineered uml class diagrams," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 396–399.

[85] X. Xiao, X. Wang, Z. Cao, H. Wang, and P. Gao, "Automatic identification of sensitive ui widgets based on icon classification for android apps," in *Proceedings of the 41st ACM/IEEE International Conference on Software Engineering*, ser. ICSE 2019, 2019.

[86] F. Huang, J. F. Canny, and J. Nichols, "Swire: Sketch-based user interface retrieval," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, 2019, p. 104.

[87] D. Zhao, Z. Xing, C. Chen, X. Xia, and G. Li, "Actionnet: vision-based workflow action recognition from programming screencasts," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 350–361.

[88] S. Yu, C. Fang, Y. Feng, W. Zhao, and Z. Chen, "Lirat: layout and image recognition driving automated mobile testing of cross-platform," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 1066–1069.

[89] A. Swearngin and Y. Li, "Modeling mobile interface tappability using crowdsourcing and deep learning," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–11.

[90] A. Yuan and Y. Li, "Modeling human visual search performance on realistic webpages using analytical and deep learning methods," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–12.

[91] Z. Wu, Y. Jiang, Y. Liu, and X. Ma, "Predicting and diagnosing user engagement with mobile ui animation via a data-driven approach," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–13.

[92] M. Leotta, A. Stocco, F. Ricca, and P. Tonella, "ROBULA+: An algorithm for generating robust XPath locators for web testing," *JSEP*, vol. 28, no. 3, pp. 177–204, 2016.

[93] ——, "Using multi-locators to increase the robustness of web test cases," in *Proceedings of 8th IEEE International Conference on Software Testing, Verification and Validation*, ser. ICST '15. IEEE, 2015, pp. 1–10.

[94] ——, "ROBULA+: An algorithm for generating robust XPath locators for web testing," *Journal of Software: Evolution and Process*, pp. 28:177–204, 2016.

[95] M. Hammoudi, G. Rothermel, and A. Stocco, "WATERFALL: An incremental approach for repairing record-replay tests of web applications," in *Proceedings of 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '16. ACM, 2016, pp. 751–762.

[96] M. Hammoudi, G. Rothermel, and P. Tonella, "Why do record/replay tests of web applications break?" in *Proc. of ICST '16*, 2016, pp. 180–190.

[97] H. Yee, S. Pattanaik, and D. P. Greenberg, "Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments," *ACM Transactions on Graphics (TOG)*, vol. 20, no. 1, pp. 39–65, 2001.

[98] B. Yang, F. Gu, and X. Niu, "Block mean value based image perceptual hashing," in *2006 International Conference on Intelligent Information Hiding and Multimedia*. IEEE, 2006, pp. 167–172.

[99] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[100] C. L. Novak and S. A. Shafer, "Anatomy of a color histogram," in *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 1992, pp. 599–605.

[101] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.

[102] I. Daubechies, "The wavelet transform, time-frequency localization and signal analysis," *IEEE transactions on information theory*, vol. 36, no. 5, pp. 961–1005, 1990.

[103] R. Yandrapally, A. Stocco, and A. Mesbah, "Near-duplicate detection in web app model inference," in *Proceedings of 42nd International Conference on Software Engineering*, ser. ICSE '20. ACM, 2020, p. 12 pages.

[104] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *EMSE*, vol. 10, no. 4, pp. 405–435, 2005.

[105] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A Database of Existing Faults to Enable Controlled Testing Studies for Java Programs," in *Proc. of the ISSTA '14*, 2014, pp. 437–440.

[106] G. Fraser and A. Arcuri, "Sound empirical evidence in software testing," in *Proc. of ICSE '12*, 2012, pp. 178–188.

[107] P. Gyimesi, B. Vancsics, A. Stocco, D. Mazinanian, A. Beszedes, R. Ferenc, and A. Mesbah, "BugsJS: a benchmark of JavaScript bugs," in *Proceedings of the International Conference on Software Testing, Verification, and Validation (ICST)*. IEEE Computer Society, 2019, p. 12 pages. [Online]. Available: /publications/docs/icst19.pdf

[108] P. Gyimesi, B. Vancsics, A. Stocco, D. Mazinanian, Árpád Beszédes, R. Ferenc, and A. Mesbah, "BugJS: A benchmark and taxonomy of javascript bugs," *Software Testing, Verification And Reliability*, 2020.

[109] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[110] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. of ICCV '01*, 2001, pp. 416–423.

[111] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 594–611, Apr. 2006. [Online]. Available: https://doi.org/10.1109/TPAMI.2006.79

Mohammad Bajammal is currently pursuing his doctoral study at the University of British Columbia (UBC), where he obtained his masters. His research interests include computer vision-oriented techniques for software testing, UI development, and program analysis for web applications. He won the Distinguished Paper Award at the 2018 International Conference on Software Testing, Verification and Validation (ICST).

Andrea Stocco is a postdoctoral fellow at the Software Institute (USI), Switzerland. His research interests include web testing and empirical software engineering, with particular emphasis on test breakage detection and automatic repair, robustness and maintainability of test suites for web applications. He is the recipient of the Best Student Paper Award at the 16th International Conference on Web Engineering (ICWE 2016). He serves on the program committees of top-tier software engineering conferences such as FSE and ICST, and reviews for numerous software engineering journals including TSE, EMSE, TOSEM, JSS, and IST.

Davood Mazinanian is postdoctoral fellow at the University of British Columbia (UBC). Prior to that, he was a research assistant at the Gina Cody School of Engineering and Computer Science, Concordia University, where he received his PhD in Software Engineering. His research interests include software maintenance and refactoring, program analysis, and empirical studies, with emphasis on web applications. He has served as a program committee member or reviewer for multiple software engineering conferences and journals, including TSE, EMSE, JSS, FSE, ICSME, and SANER.

Ali Mesbah is an associate professor at the University of British Columbia (UBC) where he leads the Software Analysis and Testing (SALT) research lab. His main area of research is in software engineering and his research interests include software analysis and testing, web and mobile-based applications, software maintenance and evolution, debugging and fault localization, and automated program repair. He has published over 60 peer-reviewed papers and received numerous best paper awards, including two ACM Distinguished Paper Awards at the International Conference on Software Engineering (ICSE 2009 and ICSE 2014). He was awarded the NSERC Discovery Accelerator Supplement (DAS) award in 2016. He is currently on the Editorial Board of the IEEE Transactions on Software Engineering (TSE) and regularly serves on the program committees of numerous software engineering conferences such as ICSE, FSE, ASE, ISSTA, and ICST.