The contents of this document are as follows:

Computer Vision in Software Engineering: A Survey

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 mainly around techniques engineering the source code. However, novel approaches to 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 paper is to provide a comprehensive survey of the current state of the art of the use of computer vision techniques in SE with the aim of assessing their potential in advancing SE research. We examined an extensive body of literature from top-tier SE venues; our inclusion criteria targeted papers applying computer vision techniques that address problems related to any area of SE. We collected an initial pool of over 1,332 papers, from which we retained 42 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 results highlight that visual approaches have been mainly used in effectively tackling software testing problems for web and mobile domains. The results of our survey show a rapid growth in the use of computer vision techniques for SE research. We also summarize the main contributions, as well as their limitations, together with some directions for future work.

Index Terms—Computer Vision, Software Engineering, Survey.

1 Introduction

Software engineering (SE) is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [1]. All areas of the software engineering lifecycle— such as 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 approaches have typically revolved around the source code, whether to improve its quality or maintainability, testing it, 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 an SE problem or task, but using visual techniques instead of relying 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 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 [2, 3, 4], to identify cross-browser incompatibilities in web pages [5, 6, 7], or to simplify software requirements modelling [8, 9].

In this paper, we surveyed the literature on the use of computer vision approaches for software engineering tasks. The aim of this work is to explore, analyze and highlight the contributions and potential of CV approaches in advancing software engineering. Our work aims to showcase 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 the use of CV in SE, building a concrete understanding of the advances made, and synthesizing insight for future research directions.

We conducted a survey by formulating a number of research questions to fulfil the goal of the study; then we 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 they brought benefit to, what CV techniques have been used, and what is the rationale for their adoption. This paper makes the following contributions:

- the first survey on the use of computer vision (CV) for software engineering, to the best of our knowledge.
- a study of the software engineering areas 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, and 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.*

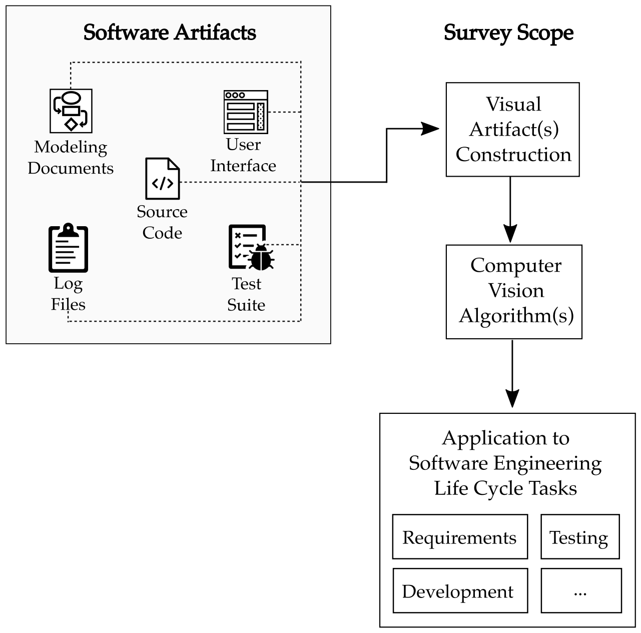

Fig. 1: Overview of the scope of this survey.

2 Methodology

In order to conduct the survey in a thorough and structured manner, we follow the established guidelines by Kitchenham et al. [10]. 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.

2.1 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 [1]. 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 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). *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 was 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.

2.2 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 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 further clarifies the scope of this work in relation to the software engineering life cycle and other software artifacts. As shown in the figure, the scope is to survey the visual artifacts and visual approaches used (as defined in section 2.1), as well as the software engineering areas and tasks where visual approaches have been used. The figure also helps clarifying what areas are outside the scope of this survey. For instance, as can be seen in the figure, the scope is not concerned with surveying typical software artifacts such as Unified Modeling Language (UML) modeling documents or log files, as there have been many surveys already on their use in software engineering [11, 12, 13]. The scope is also not concerned with cases where computer vision processing was not utilized. Further details on what is or is not included in this survey are covered in sections 2.4.1 and 2.4.1, where we provide details on our inclusion and exclusion criteria.

2.3 Research Questions

As discussed in section 2.2, 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:

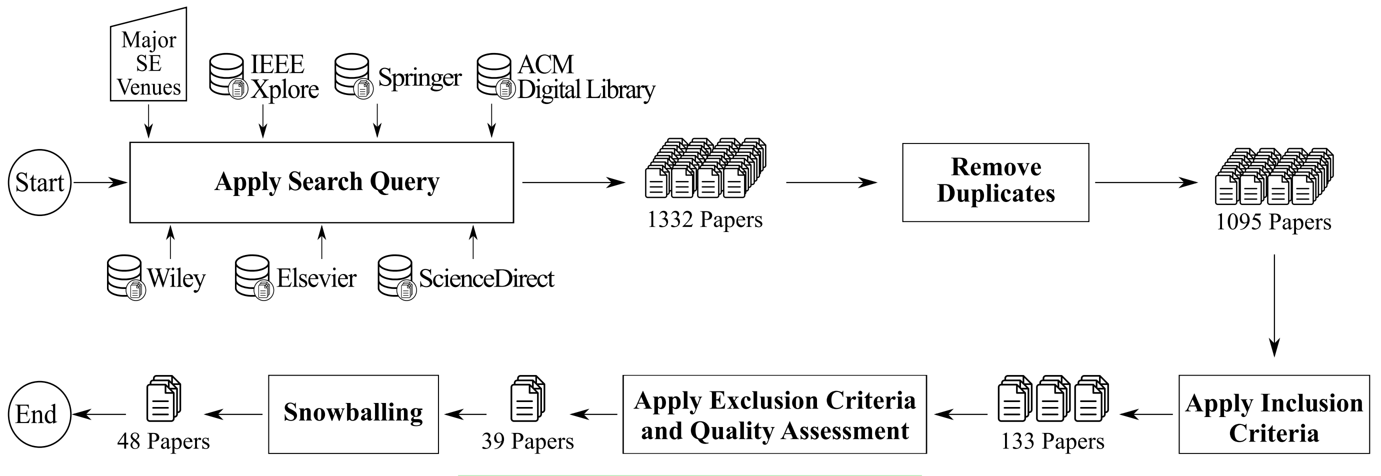

Fig. 2: Overview of the selection process.

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

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.

2.4 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, ScienceDirect, Springer, Wiley, and Elsevier. The search covers papers that have been published until March 2019 (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.

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 1,332 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.

2.4.1 *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 modelling, 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 whitepapers, 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.

We exclude papers published in the areas of software visualization or visual representation of various aspects of software for the following reasons. First, the visualization class of algorithms does not constitute a *visual approach*, as defined in section 2.1. 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, such as surveys on visualization for software security [14, 15], surveys on visualization for static analysis [16], development coordination [17], maintenance and evolution [18, 19], to name a few.

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.

2.4.2 *Snowballing*

At the end of searching database repositories and conference proceedings and journals, and applying inclusion,

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

	Acronym	Venue
Conferences	ICSE	International Conference on Software Engineering
	FSE	ACM SIGSOFT International Symposium on Foundations of Software Engineering
	ESEC/FSE	ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering
	ASE	IEEE/ACM International Conference on Automated Software Engineering
	ESEM	ACM/IEEE International Symposium on Empirical Software Engineering and Measurement
	ICST	IEEE International Conference on Software Testing, Verification and Validation
	ISSTA	ACM SIGSOFT International Symposium on Software Testing and Analysis
	MSR	International Conference on Mining Software Repositories
	RE	IEEE International Requirements Engineering Conference
	ICSME	IEEE International Conference on Software Maintenance and Evolution
	MODELS	IEEE/ACM International Conference on Model Driven Engineering Languages and Systems
	ISSRE	IEEE International Symposium on Software Reliability Engineering
	EASE	Evaluation and Assessment in Software Engineering
Journals	TSE	IEEE Transactions on Software Engineering
	EMSE	Empirical Software Engineering
	TOSEM	ACM 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

exclusion, and quality criteria, we obtained a total of 38 unique papers. Next, to mitigate the risk of omitting relevant literature from this survey, we also performed backward snowballing [20] by inspecting the references cited by the collected papers so far. Eight additional papers were retrieved during this phase, which led to a final pool of 42 unique papers. Table 3 shows the final pool of papers that will be discussed and analyzed in the remainder of this work.

2.4.3 *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. A list of stated reasons for adopting computer vision were 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.

3 Findings

3.1 Trends and Landscape

At the end of the paper collection process, we obtained a pool of papers spanning the years 2010-2019 (March 2019, as of this writing). Table 3 shows the final list of collected papers in our pool. Our pool consists of 31 conference papers (74%) and 11 journal papers (26%). Figure 3 shows the distribution of the collected pool of papers across different years of publication. Overall, we can observe a generally increasing trend in the use of computer vision approaches in software engineering. Furthermore, Figure 4 depicts the

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

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

Reference	Title	Venue	Year
Caetano et al. [21]	JavaSketchIt: Issues in Sketching The Look of User Interfaces	AAAI	2002
Coyette et al. [22]	Multi-fidelity Prototyping of User Interfaces	INTERACT	2007
Chang et al. [2]	GUI Testing Using Computer Vision	CHI	2010
Choudhary et al. [23]	WEBDIFF: Automated Identification of Cross-browser Issues in Web Applications	ICSM	2010
Li et al. [8]	FrameWire: A Tool for Automatically Extracting Interaction Logic from Paper Prototyping Tests	CHI	2010
Delamaro et al. [24]	Using Concepts of Content-based Image Retrieval to Implement Graphical Testing Oracles	STVR	2011
Dixon et al. [25]	Content and Hierarchy in Pixel-based Methods for Reverse Engineering Interface Structure	CHI	2011
Seifert et al. [26]	Mobidev: a tool for creating apps on mobile phones	MobileHCI	2011
Choudhary et al. [27]	Crosscheck: Combining Crawling and Differencing to Better Detect Cross-browser Incompatibilities in Web Applications	ICST	2012
Scharf and Amma [9]	Dynamic Injection of Sketching Features Into GEF-based Diagram Editors	ICSE	2013
Alégroth et al. [3]	JAutomate: A Tool for System- and Acceptance-test Automation	ICST	2013
Semenenko et al. [5]	Browserbite: Accurate Cross-Browser Testing via Machine Learning over Image Features	ICSM	2013
Roy Choudhary et al. [6]	X-PERT: Accurate Identification of Cross-Browser Issues in Web Applications	ICSE	2013
Lin et al. [4]	On the Accuracy, Efficiency, and Reusability of Automated Test Oracles for Android Devices	TSE	2014
Mahajan and Halfond [28]	Finding HTML Presentation Failures Using Image Comparison Techniques	ASE	2014
Amalfitano et al. [29]	Towards Automatic Model-in-the-loop Testing of Electronic Vehicle Information Centers	WISE	2014
Selay et al. [7]	Adaptive Random Testing for Image Comparison in Regression Web Testing	DICTA	2014
Nguyen and Csallner [30]	Reverse Engineering Mobile Application User Interfaces with REMAUI	ASE	2015
Burg et al. [31]	Explaining Visual Changes in Web Interfaces	UIST	2015
Mahajan and Halfond [32]	Detection and Localization of HTML Presentation Failures Using Computer Vision-Based Techniques	ICST	2015
Deka et al. [33]	ERICA: Interaction Mining Mobile Apps	UIST	2015
Ponzanelli et al. [34]	Too Long; Didn't Watch! Extracting Relevant Fragments from Software Development Video Tutorials	ICSE	2015
Hori et al. [35]	An Oracle based on Image Comparison for Regression Testing of Web Applications	SEKE	2015
Mahajan et al. [36]	Using Visual Symptoms for Debugging Presentation Failures in Web Applications	ICST	2016
Feng et al. [37]	Multi-objective Test Report Prioritization Using Image Understanding	ASE	2016
Patrick et al. [38]	Automatic Test Image Generation Using Procedural Noise	ASE	2016
Deka et al. [39]	Rico: A Mobile App Dataset for Building Data-Driven Design Applications	UIST	2016
Wan et al. [40]	Detecting Display Energy Hotspots in Android Apps	STVR	2016
He et al. [41]	X-Check: A Novel Cross-browser Testing Service Based on Record/Replay	ICWS	2016
Bao et al. [42]	Extracting and Analyzing Time-series HCI Data from Screen-captured Task Videos	EMSE	2017
Zhang et al. [43]	Sketch-guided GUI Test Generation for Mobile Applications	ASE	2017
Chen et al. [44]	UI X-Ray: Interactive Mobile UI Testing Based on Computer Vision	IUI	2017
Reiss and Miao [45]	Seeking the User Interface	ASE J.	2017
Kıraç et al. [46]	VISOR: A Fast Image Processing Pipeline with Scaling and Translation Invariance for Test Oracle Automation of Visual Output Systems	JSS	2017
Leotta et al. [47]	Pesto: Automated Migration of DOM-based Web Tests Towards the Visual Approach	STVR	2018
Bajammal and Mesbah [48]	Web Canvas Testing through Visual Inference	ICST	2018
Xu and Miller [49]	Cross-Browser Differences Detection Based on an Empirical Metric for Web Page Visual Similarity	TOIT	2018
Kuchta et al. [50]	On the Correctness of Electronic Documents: Studying, Finding, and Localizing Inconsistency Bugs in PDF Readers and Files	EMSE	2018
Bao et al. [51]	VT-Revolution: Interactive Programming Video Tutorial Authoring and Watching System	TSE	2018
Moran et al. [52]	Automated Reporting of GUI Design Violations for Mobile Apps	ICSE	2018
Moran et al. [53]	Machine Learning-Based Prototyping of Graphical User Interfaces for Mobile Apps	TSE	2018
Stocco et al. [54]	Visual Web Test Repair	FSE	2018
Tanno and Adachi [55]	Support for Finding Presentation Failures by Using Computer Vision Techniques	ICST	2018
Bajammal et al. [56]	Generating Reusable Web Components from Mockups	ASE	2018
Moran et al. [57]	Detecting and Summarizing GUI Changes in Evolving Mobile Apps	ASE	2018
Natarajan and Csallner [58]	P2A: A Tool for Converting Pixels to Animated Mobile Application User Interfaces	MOBILESoft	2018
Xiao et al. [59]	Automatic Identification of Sensitive UI Widgets based on Icon Classification for Android Apps	ICSE	2019
Huang et al. [60]	Swire: Sketch-based User Interface Retrieval	CHI	2019

cumulative number of publications per software engineering area across different years. 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 International Conference on Automated Software Engineering (ASE) with seven (7) papers, the International Conference on Software Testing, Validation and Verification (ICST), with six (6) papers, and the International Conference on Software Engineering (ICSE) with five (5) papers.

3.2 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 benefited the most from visual approaches, for which *tasks*, and on which *platforms* they were used. As defined in section 2.1, 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 used (e.g., desktop, mobile). 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.

3.2.1 *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 by far the most common research area for which approaches using computer vision are proposed, accounting for approximately 75% of 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 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. [50] 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. [47] propose an automated code migration tool for automatically converting end-to-end Selenium web tests to visual web tests based on Sikuli's [2] image recognition capabilities. Kıraç et al. [46] 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 [48] 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.

A significant class of software testing techniques leveraging visual methods aims at automatically identifying *cross-browser incompatibilities* (XBIs) for web applications [5, 6, 7, 23, 27, 41, 49]. XBIs are frequently-occurring issues in web pages' appearance and/or behaviour when the same page is viewed on different web browsers [6]. Identifying such differences requires laborious human judgment, which can be effectively reduced using an automated visual-based technique. A recent SRL by Sabaren et al. [61] 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 [28, 32, 36]. In addition to the web domain, root cause analysis is also used to identify rendering issues in PDF files [50], as well as, the causes of excessive energy consumption of UIs in mobile applications [40].

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 [2, 3, 4, 41] . Popular visual record-and-replay tools are Sikuli [2] and JAutomate [3], 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. [29] propose a tool to

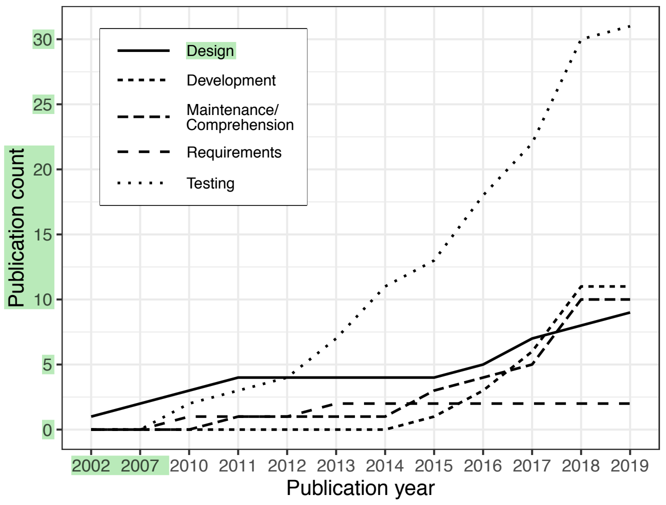

Fig. 4: Cumulative distribution of publications across years per SE area. Testing is the most common area, followed by development and maintenance.

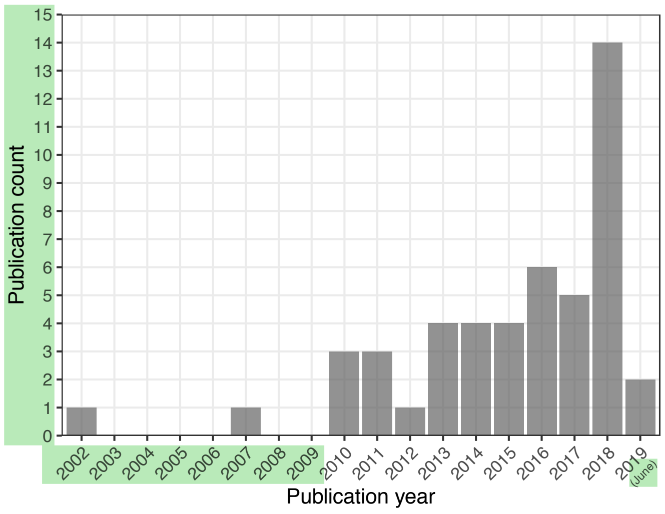

Fig. 3: Distribution of publications across years.

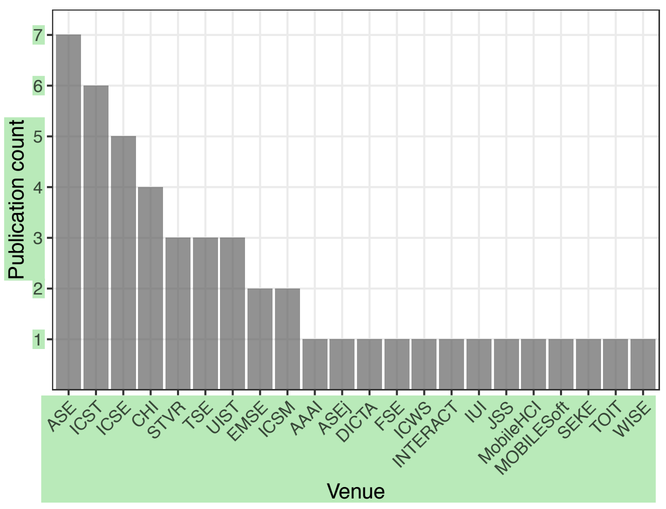

Fig. 5: Distribution of the publications across venues.

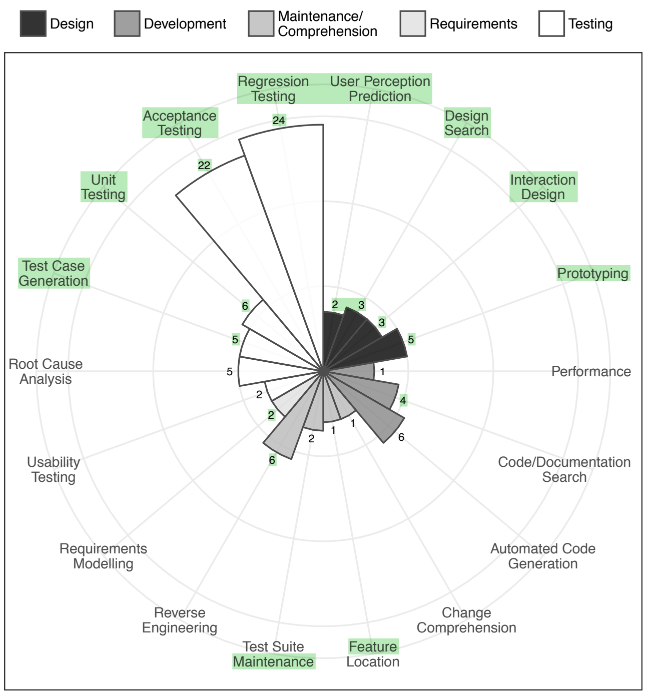

Fig. 6: Papers distribution across different Software Engineering Areas and Tasks

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.

Patrick et al. [38] 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. [37] 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. [43] 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 revealed only a few works aiming at facilitating the design process of software systems by means of computer vision. Li et al. [8] propose a tool to help with *prototyping* software designs. Using computer vision techniques, a video recording of paper-based 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. Deka et al. [33] 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. [39] 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.

Requirements Engineering. Requirements engineering was the least explored area among all collected papers, with only two (2) 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 [9], or by augmenting existing requirement artifacts to make them user-tractable [8].

Comprehension and Maintenance. Visual techniques have been used in software *reverse engineering*. The REMAUI tool [30], for example, 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.

In another work, Burg et al. [31] 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*). Stocco et al. [54] present a visual approach for test maintenance, where they address the issue of web test breakage by proposing a technique to repair web test cases by visual analysis of test execution. Finally, Leotta et al's approach [47] 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*, facilitating the development of software systems. This includes generating UI code from mockups [30, 56], existing mobile apps UI code [33, 39], or hand-made sketches [45].

Reiss and Miao [45] 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. [34] allow searching relevant code fragments from video tutorials using visual techniques. Bajammal et al. [56] generate UI component code (e.g., React, AngularJS) from a visual analysis of a web app's mockup design. Finally, Wan et al. [40] allow to spot energy pitfalls in the UIs of mobile apps, allowing a more performance-aware UI development.

3.2.2 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 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 [29, 46], e.g., automotive 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 visual-like artifact.

TABLE 4: Papers distribution across different platforms

Platform	Studies
Web	[5, 6, 7, 8, 23, 27, 28, 31, 32, 35, 36, 41, 47, 48, 49]
Desktop	[2, 3, 9, 24, 34, 37, 38, 42, 45, 50]
Mobile	[4, 30, 33, 39, 40, 43, 44]
Other	[29, 46]

3.3 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, we analyzed the text and noted down the rationales mentioned by the authors for using computer vision to solve the software engineering problem being tackled. After conducting this process over all collected papers, we grouped the rationales and identified three main patterns of motivations, which we now describe.

3.3.1 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 [2, 3, 7, 8, 9, 24, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 42, 44, 45, 46, 47, 48, 49, 50].

For instance, Chang et al. [2] 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.

3.3.2 Ease of Use

As a second main motivation, researchers have utilized visual approaches because they deemed them *easier* to use by end users [4, 5, 6, 7, 23, 24, 27, 28, 29, 32, 36, 41, 43, 44, 49, 50]. For instance, Zhang et al. [43] 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) [5, 6, 7, 23, 27, 41, 49]. 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.

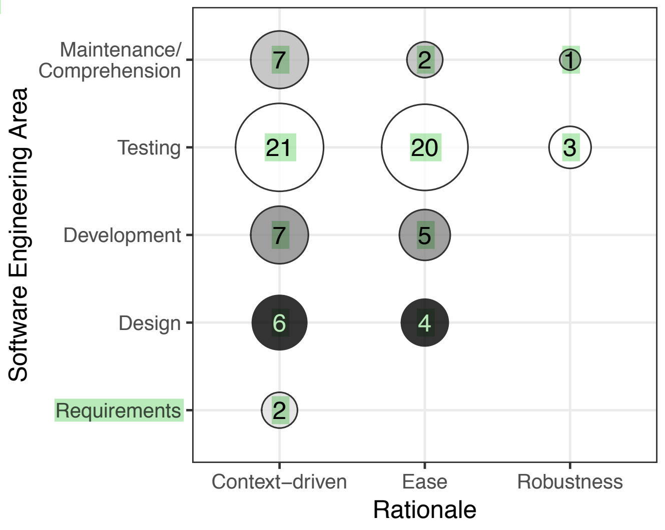

Fig. 7: Distribution of rationales per SE area.

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

3.3.3 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 [2, 3, 55] 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. [2] describe how spatial rearrangements of GUI components on the page can lead to fragile test scripts, a well-known issue in web testing [62]. According to Chang et al. [2], 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. [3], 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. [54], 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 researchers [63, 64, 65, 66].

The bubble chart of Figure 7 shows the distribution of the rationales in relation to the SE areas presented in Section 3.2. 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. [43], 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.

3.4 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.

3.4.1 *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) [5, 6, 23, 24, 27, 30, 32, 33, 35, 36, 37, 38, 39, 40, 41, 43, 44, 46, 49, 50]. 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 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 [2, 3, 7, 28, 29, 31, 47, 48]. 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

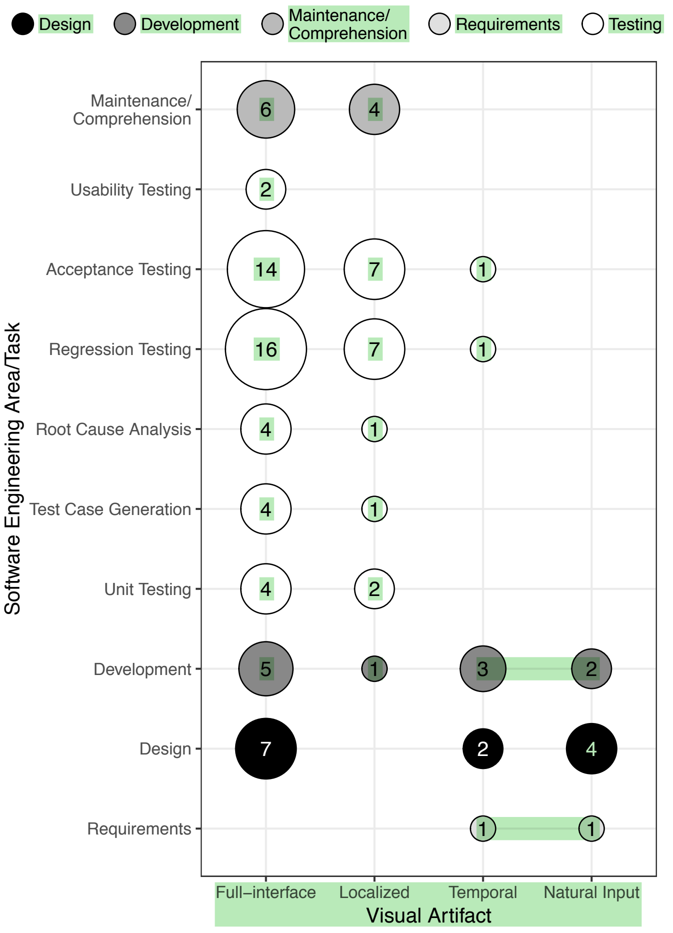

Fig. 8: Distribution of visual artifacts per SE area & task.

artifacts have been used to create a test case for a GUI (by recording and tracking a single visual artifact for UI elements) [2], or in debugging the rendering or capturing the behaviour of a specific HTML element in a web application [31].

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 [4, 8, 34, 42]. For instance, Bao et al. [42] 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 [9, 45]. 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.

TABLE 5: Taxonomy of major computer vision algorithms synthesized from the survey.

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 [67], PHash–Perceptual Hashing [68]).	[8, 23, 28, 31, 32, 34, 36, 39, 42, 44, 46, 49, 52, 57]
	Probability distribution distance	Measuring the distance between *distributions* of pixels in a pair of images (e.g. χ^2 distribution distance).	[4, 6, 23, 27, 35, 37, 41, 49, 56, 57]
Transformational	Color/Spatial transformation	Applying a transformation matrix to the spatial or color-space of one or more images.	[8, 38, 40, 46, 48, 52, 53, 59]
	Optical character recognition	Recognizing images of strings and converting them into textual data.	[2, 29, 30, 34, 51, 55, 59]
Search	Template matching	Finding where an image is located within another image.	[2, 4, 5, 42, 44, 47, 54, 55]

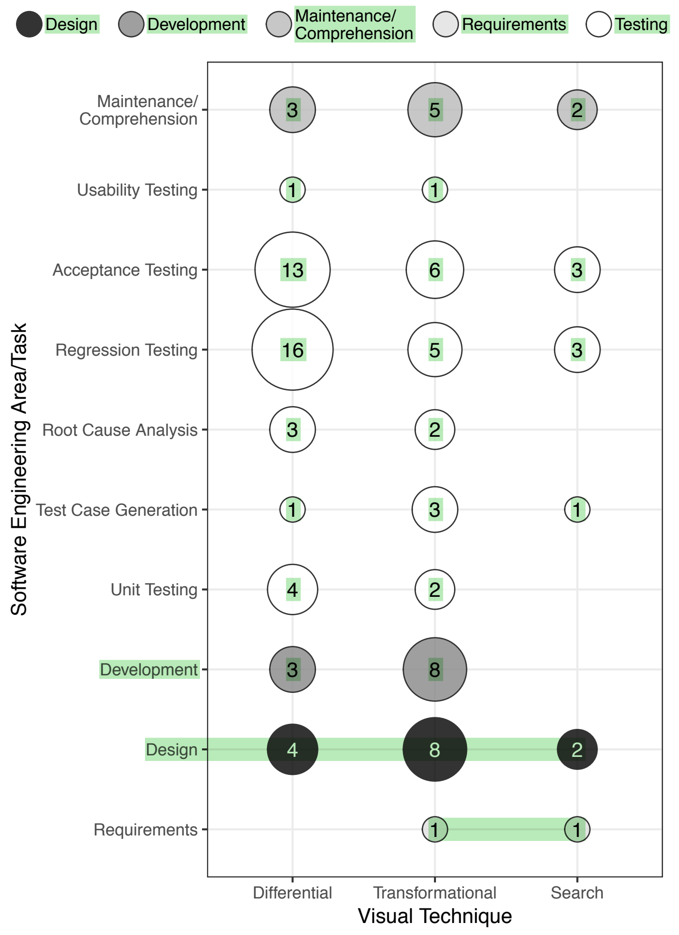

Fig. 9: Distribution of visual techniques per SE area & task.

3.4.2 Visual Technique Categories

We identified three main categories of visual techniques used in software engineering research: (1) differential, (2) transformational, and (3) search techniques.

Differential techniques are utilized to have two or more visual artifacts contrasted and their differences extracted [3, 4, 6, 7, 23, 24, 27, 28, 29, 31, 32, 35, 36, 37, 39, 41, 46, 49]. 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 [7, 23, 27, 41, 49], 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 [5, 9, 30, 33, 34, 38, 40, 42, 43, 45, 48, 50]. 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 [9], where transformational techniques are applied on hand-drawn stroke instances to extract testing instructions and assertions from the strokes, and finally generate a working test case.

Finally, the third category is *search techniques* [2, 8, 44, 47]. 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 [2]. 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 9 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.

3.4.3 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 [8, 23, 28, 34, 39, 42, 44, 46, 52], 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) [67], PHash (Perceptual Hashing) [68], and Structural Similarity Index (SSIM) [69], which aim to reduce false positives by mimicking human perception. These techniques were utilized in [31, 32, 36, 49, 52, 57].

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* [70] 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 [6, 27, 35, 37, 41, 57], 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 [8, 38, 40, 46, 48, 52, 53, 59]. 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 [2, 4, 5, 42, 44, 47, 54, 55] 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 [2, 29, 30, 34, 51, 55, 59] 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 [5, 27]. *Convolutional neural networks* (CNNs) were also used to analyze GUIs and their content [5, 53]. Furthermore, scale- and transformation-invariant features (e.g. SURF–Speeded-Up Robust Features [71], Wavelets [72]), which are basically features aiming at analyzing image structure and content, were used to classify and detect GUI elements [42, 50, 54, 59].

3.4.4 Libraries

We also investigated what libraries or frameworks were used by the papers in our pool in their implementation of CV techniques. 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 3.4.3). It was first released in 2000, and is still in active development, with the latest release (as of this writing) in December of 2018.

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). *Abby FineReader* [3] focuses specifically on OCR and related preprocessing/postprocessing algorithms.

3.5 Evaluation and Challenges (RQ4)

3.5.1 Evaluation Methods

We systematically studied the selected papers to understand the most common methods used for evaluating the proposed visual approaches. Table 6 (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 [4, 6, 23, 27, 30, 32, 35, 37, 38, 40, 41, 42, 44, 45, 46, 48, 49, 50].

Another significant body of work measured the manual effort saved by the visual approach with respect to the humanly performed task [5, 8, 8, 24, 32, 37, 44, 45, 47]. 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) [7, 30, 35, 38, 40, 42, 45, 46].

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 favour 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

1. https://opencv.org
2. https://imagemagick.org/
3. https://www.abbyy.com/

TABLE 6: Evaluation Methods and Challenges

	Studies
Evaluation Methods	
Accuracy Measures	[4, 6, 23, 27, 30, 32, 35, 37, 38, 40, 41, 42, 44, 45, 46, 48, 49, 50]
Comparison against manual work	[5, 8, 8, 24, 32, 37, 44, 45, 47]
Survey or manual judgment/validation of the results	[6, 23, 27, 32, 34, 35, 36, 49, 50]
Time Comparison	[7, 30, 35, 38, 40, 42, 45, 46]
Comparison against competitor approaches	[3, 4, 36, 39, 41, 43, 49]
Comparison to original design/output	[30, 38, 40, 47, 48]
No real evaluation/only use case is shown	[3, 9, 31, 33]
Industry context/feedback	[8, 29, 32, 50]
Comparison with random/brute force approach	[7, 28, 37, 38]
Seeded Faults	[28, 32, 36, 38]
Comparison against different versions of the input	[2, 36]
Line/state coverage	[39, 43]
Comparison on different physical devices	[4]
Challenges	
Noise	[4, 8, 30, 34, 42, 45, 46, 47, 50]
Dynamicity	[2, 23, 38, 40, 48]
Recognition	[9, 42, 46, 50]
Visibility	[2, 23, 42, 47]

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 [6, 23, 27, 32, 34, 35, 36, 49, 50]. As an example, Mahajan and Halfond [32] 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 rigid 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 [3, 9], where having a comprehensive evaluation is not mandatory. The other two papers [31, 33] were published at the ACM Symposium on User Interface Software and Technology (UIST), in which the evaluation methods are more light-weight than most of the top-tier software engineering venues such as ICSE.

Only four works included an industrial evaluation [8, 29, 32, 50]. This is unfortunate, since the ultimate goal of any software engineering techniques is to demonstrate their industrial applicability to solve real-world problems at scale.

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 [28, 32, 36, 38] use “seeded faults” as an approach for constructing test oracles, whereas others take a different evaluation path. For instance, Roy Choudhary et al. [6] manually count all XBIs detected by different tools (i.e., their agreement) as an upper bound for 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.

3.5.2 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 6 (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 [4, 8, 30, 34, 42, 45, 47, 50]. 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. [34] 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 [2, 4, 8, 46, 47, 50]. 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 [2, 23, 48]. 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 advertisements [40] (also referred to as *ads*). This form of marketing is commonly being utilized in web sites, and they have even overtaken traditional cable TV last year [73]. An ad is realized 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 [46], 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. [42] discuss problems in detecting visual fine-grained developers' actions from videos, such as code editing, text selection, and window scrolling. Similarly, Scharf and Amma [9] 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* [2, 47] and *scrolling* [23, 42].

As an example, Leotta et al. [47]'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.

4 Discussion

Increasing Adoption of Visual Approaches. Our findings show a general growth in the use of visual approaches in the SE community. Most of the surveyed works 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, most of software developed nowadays has 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.

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. [23] use an image comparison measure based on Earth Mover's Distance (EMD), whereas Mahajan and Halfond [32] adopt perceptual differencing, and He et al. [41] 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 [74], Defects4J [75], SF100 [76], and BugsJS [77]. 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 [78], BSDS500 [79], or Caltech 101 [80]. 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.

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. [3] 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. [24] 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.

4.1 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 2.4). 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 3). 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.

5 Related Work

To the best of our knowledge, there are no surveys or systematic literature reviews (SLRs) examining the literature using computer vision for software engineering. However, a few SLRs have been conducted in areas of software testing where some visual approaches were used, which we are going to discuss in this section.

Issa et al [81] 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. [82, 83] present a case study of the benefits and challenges of using visual GUI testing by the team at one software company. Garousi et al. [84] compare two popular visual testing tools (Sikuli and JAutomate) in one industrial project, and go through differences in test creation process, execution, and maintenance. Leotta et al. [85] perform an empirical examination of visual against non-visual (e.g., DOM) web testing locators. They compare the robustness of the locators and the robustness for evolving software, and describe the pros and cons of each approach.

The closest work to our survey is the SLR by Sabaren et al. [86] They present an SLR in the context of cross-browser testing (i.e., XBI testing). According to the findings, the most used technique is visual analysis. They report that

55% of the analyzed pool of papers use computer vision. 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 XBIs detection techniques, causing many false positives in practice. Their work differs from our survey in a number of ways. First, our scope covers all areas of software engineering (e.g., requirements, design, testing) whereas the scope in Sabaren et al. [86] is specific to the task of cross-browser testing. In addition, their work does not focus specifically on computer vision, while this survey aims at exclusively investigating computer vision techniques. Nevertheless, this survey has also shown that visual techniques have been successfully applied to detecting XBIs, and our pool shares 11 papers about XBIs with the visual-based papers reported in the SLR by Sabaren et al. [5, 6, 7, 23, 27, 28, 32, 35, 36, 41, 46].

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 1,332 publications, we systematically selected 42 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] I. C. Society, P. Bourque, and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK)*, 3rd ed., 2014.

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

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] B. Kitchenham and S. Charters, “Guidelines for performing systematic literature reviews in software engineering,” 2007.

[11] H. Mumtaz, M. Alshayeb, S. Mahmood, and M. Niazi, “A survey on uml model smells detection techniques for software refactoring,” *Journal of Software: Evolution and Process*, vol. 31, no. 3, p. e2154, 2019.

[12] A. M. Fernández-Sáez, D. Caivano, M. Genero, and M. R. Chaudron, “On the use of uml documentation in software maintenance: Results from a survey in industry,” in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. IEEE, 2015, pp. 292–301.

[13] Y. D. Salman and N. L. Hashim, “Automatic test case generation from uml state chart diagram: a survey,” in *Advanced Computer and Communication Engineering Technology*. Springer, 2016, pp. 123–134.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] S. R. S. Choudhary, H. Versee, and A. Orso, "WEBDIFF: Automated Identification of Cross-browser Issues in Web Applications," in *Proc. of ICSM '10*, 2010, pp. 1–10.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

[29] 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.

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

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

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

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

[46] 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.

[47] 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.

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

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] L. N. Sabaren, M. A. Mascheroni, C. L. Greiner, and E. IrrazÃ¡bal, "A systematic literature review in cross-browser testing," *JCST*, vol. 18, no. 01, April 2018.

[62] 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.

[63] ——, "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.

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

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

[70] 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*, June 1992, pp. 599–605.

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

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

[73] "Super bowl ads can't save tv." http://fortune.com/2018/01/23/super-bowl-2018-commercials-ads-tv/, 2018.

[74] 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.

[75] 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.

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

[77] 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

[78] 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.

[79] 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.

[80] 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

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] M. Leotta, D. Clerissi, F. Ricca, and P. Tonella, "Visual vs. dom-based web locators: An empirical study," in *International Conference on Web Engineering*. Springer, 2014, pp. 322–340.

[86] L. N. Sabaren, M. A. Mascheroni, C. L. Greiner, and E. Irrazábal, "A systematic literature review in cross-browser testing," *Journal of Computer Science and Technology*, vol. 18, no. 01, April 2018. [Online]. Available: http://journal.info.unlp.edu.ar/JCST/article/view/691

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 currently a postdoctoral fellow at Università della Svizzera Italiana (USI). Before that, he was a postdoctoral fellow at the University of British Columbia (UBC), Canada. He received his PhD in Computer Science at the University of Genova, Italy, in 2017. He is the recipient of the Best Student Paper Award at the 16th International Conference on Web Engineering (ICWE 2016). 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.

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.

Reviewer #1

- There were a couple aspects of the authors presented literature review methodology that were unclear to me. The authors state that they used several databases of well-known publishers of scientific literature (IEEE Xplore, ACM Digital Library, etc.), however they did not specify the publication venues upon which they were focusing their paper collection. Even so, they managed to only collect papers from generally well-known software engineering and HCI venues. From recent experience with the databases mentioned in the paper, conducting a search without limiting to certain publication venues can result in the inclusion of lower quality work from less well-known venues. The authors should describe their process for filtering publication venues or explain how their inclusion/exclusion criteria led to the collected papers coming from the venues specified in the paper.
- Thank you for pointing this out. We do specify the publication venues we use (e.g. Table 1). We explain in the paper that we use both, publication databases as well as specific publication venues, to collect papers. Our inclusion and exclusion criteria would then focus on building a pool of papers with detailed information that we can use to answer our research questions. We will clarify this further in the revised version.

- As part of the author's exclusion criteria, they excluded papers published in the areas of software visualization or visual representations of various aspects of software engineering. While at some level, I understand this exclusion (software visualization papers could have their own survey), the authors should more clearly describe this scope in the introduction and abstract of the paper. Furthermore, there many be work at the intersection of software visualization and software engineering where visual artifacts are used as input to help improve some form of visualization. The authors should comment on these types of papers, if they came across when applying the inclusion/exclusion criteria, and justification for why such papers were not included.
- Thank you for the great comment. We will clarify this further in the scope and introduction of the manuscript. First, while analyzing the pool of papers, we have not found papers that were at the intersection of both approaches (i.e., used visual artifacts

as input and targeted a software visualization problem). Second, we note that the reason for excluding software visualization is that, when a keyword search is performed using terms such as “visual” and “software”, the vast majority of retrieved results were software visualization papers. Therefore, we had to specify visualization as an exclusion criteria in order to be able to retrieve papers that focused on computer vision.

- Across many of the paper’s analyses and tables, it seems that there are a handful of recent papers that are excluded from the analysis. For example, Table 4 does not list any paper after reference 37. This is also the case for Table 6. The authors should be sure that an analysis of all papers was included for analyses throughout the paper.
- Thanks for pointing that out. It was a typo because some papers were added at a later stage. This will be fixed in the revised version.

- Overall, I am satisfied with the authors characterizations and answers to the research questions posed at the beginning of the paper, and I believe the research questions do a fair job of distilling important aspects of work conducted related to computer vision in software engineering. However, this being said, the authors present a large amount of information in response to each research question, and I found it difficult to distill the most important pieces from the prose at the end of different subsections. I would suggest that the authors add an executive summary of the findings for each research question following each subsection discussing the findings.
- That’s a great suggestion. We will add an executive summary at the end of each research question.

- One somewhat minor point is that it would be appreciated if the authors could make the data extraction and classifications of the papers available in an online appendix or GitHub repository. This would help to make all of the work that went into collecting this information more actionable for the general research community.
- Yes for sure, we will make it available online.

Reviewer #2

- I find both the (clarity of) writing/reporting, the motivation of the study, and the actual litrev/survey methodology employed quite problematic. At least these aspects need to be clarified extensively. I'm also not convinced the authors have considered/found all relevant papers. However, I do see value in getting a more coherent view of computer vision (CV) in SE and some kind of path forward for the possibilities of this field. Even though I foresee quite major revisions needed, potentially even including additional paper searching, screening and extraction, I'm willing to give the authors a chance to do this given the potential value of the end result. I thus recommend major revision rather than rejection. The main things that the revision should address/focus on:

  - Scoping of the paper.

  - Search (possibly updated search, screening and extraction) that is clearly in line with the scoping.

  - More synthesis and analysis when summarising the final pool of papers. Possibly adding some kind of taxonomy to guide future work in more detail? Or some kind of guideline for CV in SE research/applications? Personally, I think a plain sysrev is no longer enough and as a community we should aim higher. I understand this might not be a majority viewpoint though (so the editor will have to decide what level he/she requires here). (If not going for the taxonomy then adding more info from the CV research area itself might be another way to add more value to SE; see below for more details on this)

- Thank you for your valuable feedback. We do address the scoping of the paper in its own separate subsection (Section 2.2). We will clarify it further and flesh out the scope in more detail. As for taxonomy, we do classify techniques at both coarse and fine-grained levels as in Fig. 9 and Table 5. However, to clarify this further, we will add more details in section 3.4 and better contrast the different classes.

Weaknesses:

- Title: Title too ambiguous, there are many possible aspects of / levels at which computer vision can be used in SE and/or SE used in computer vision. I appreciate the difficulty of finding an apt title for this study but currently the title does not give me enough guidance on the scope of the paper.

  - Our goal is to cover areas of software engineering in which computer vision has been used, hence the title “Computer Vision in Software Engineering: A Survey”. We do not focus on a specific level or aspect (e.g., software testing)

- Unclear methodology (early on): Not clear early on what type of survey this is. The term has been used in many and varying meanings and it would help the reader if you make it more clear already in intro and/or abstract what type of survey this is. You say it is systematic which might hint at SLR or Sysmap but this is not clear after reading the intro.

  - The term systematic was used to indicate a structured and ordered process of conducting the research, not to hint that the paper is an SLR. We will however clarify this further early on in the introduction.

- Not all contributions follow from the introduction. For example, "a study of SE areas benefiting from CV" has not been hinted at in the intro, neither why it is needed or by which method you have done this etc. This goes for several of the contributions. Expanding the intro somewhat to clarify what you did in the survey (and why) would help.

  - Thank you for the suggestion. We will further elaborate this in the introduction.

- Similarly, the RQs doesn't clearly follow from the introduction/motivation. RQs mostly make sense to me but without my own background in the area I'm not sure they would be as natural and I do think the general SE reader would need more discussion and moatvation for them. This is connected to the contributions so clarify these aspects as a whole.

  - This will be further motivated in the intro, together with the contributions in the previous comment.

- Concerns about the initial search process: See more below.

- Scoping of paper not clear: See more below.

- Reliability of screening unclear: No inter-rater agreement analysis was done among authors in the screening step. This is quite standard in modern-day SLR studies and the lack of it indicates a possibly ad hoc method / study process which does not build confidence.

- Thank you for your valuable comment. Inter-rater analysis is not standard. For instance, of the last 20 surveys and SLRs published in TSE, only one conducted inter-rater analysis. Our methodology is therefore in-line with commonly accepted screening procedures.

- Motivation for choices and connections between parts of study/paper lacking: Why did you extract the information items you extracted? How did you arrive at them from your goals and RQs? etc. I lack clear connections in the paper from earlier choices to later ones in the study and in its reporting. This lessens the value of the paper.

- Thanks for pointing this out. We will elaborate on the motivation and rationale of data extraction, and better link it to the RQs.

- Taxonomy is hidden in the paper and not well motivated: The categories you use in answering your different RQs constitutes a kind of taxonomy. However, it is not clear how you arrived at these categories and why they are the main ones. Sometimes it is not even clear how the categories differ (like in 3.4.1). Making the taxonomy more explicit and its categories more clear, distinct, and motivated would make the value of the paper more clear and it would be more useful for the community.

- Thank you for the valuable comment. We will clarify this further by adding more details in section 3.4 to better contrast the different classes and flesh out the motivations.

- CV algorithm/approach "map" is missing: You base your description of CV algs on what you found in the papers and not on the field of CV itself. Adding the latter would enable you to "diff" the current state-of-the-art in SE with that in CV. At least on a kind of "CV textbook" level this might be doable and, potentially, very valuable. This is somewhat complementary to the taxonomy proposal above.

- Thank you for your valuable suggestion. Summarizing the field of CV itself is outside the scope of this work and can not be summarized in a single manuscript. The field of CV has hundreds of algorithms for a vast number of problem categories, such as robotics and 3D cameras. Delineating, filtering/grouping, or classifying these is outside the scope of the manuscript.

- Unclear difference to related work: You mention some related work but don't clarify how it is different from yours. Doing so would clarify your contributions.

- Thanks for pointing this out. We will add more details to better flesh out the differences with respect to related work.

Major comments:

MA1. Concerns about the initial search process or at least how you describe it.

- It seems strange that, for example, a search for ("visual" AND "testing") in the text of papers would not return more than 1332 papers. It would give a lot of non-relevant papers probably but "visual" is such a general term.

- Also there seems to be a mismatch between the manual screening of recent major SE venues and the searching of the text of the papers that you could do in the databases. I assume you mean that you first screened papers from the conferences by title etc and only later search their full-body text? More details are needed around this step.

- Thank you for your comment. We are not sure what is the mismatch you are referring to. We use two categories of sources for paper collection: 1) databases, in order to maximize collection of all potentially relevant work, and 2) manual search of conferences and journals from top-tier software engineering venues, in order to double check and ensure that these top-tier venues have been specifically included since they represent the state-of-the-art work in software engineering. We will clarify this further in the manuscript.

- In general, there has been several guidelines published in SE about the construction of search querys for SLRs/Sysmap, but you show no knowledge about them and doesn't seem to have used any process for search query construction. This is a downside to the study. Consider describing more details about what you did and argue for and give evidence for why this is a "good" search string given your goals and RQs.

- Thank you for your comment. We are not sure what is meant here by query guidelines and, to the best of our knowledge, there is no single or accepted standard structure for constructing queries. In section 2.4, we do describe the terms we use for the query, why we selected them, and the logical boolean relations between the terms. Furthermore, the way we describe and construct our query is identical to the way queries are constructed in survey manuscripts. Consider, for example, the following TSE surveys and SLRs, all of which construct their queries in the same way as done in our manuscript:
    - Gazzola L, Micucci D, Mariani L. Automatic software repair: A survey. IEEE Transactions on Software Engineering. January 2019; Vol. 45, No. 1.
    - Hosseini S, Turhan B, Gunarathna D. A systematic literature review and meta-analysis on cross project defect prediction. IEEE Transactions on Software Engineering. 2017 Nov 7;45(2):111-47.

  Nonetheless, we will elaborate on this and provide more discussion in our revised manuscript.

MA2. Scoping of paper is not fully clear and thus also not its connection to inclusion/exclusion criteria.

- For example, incl criteria (2) mentions "the paper's proposed approach" which implies that only "technical" papers that propose new approaches are relevant. This is not consistent with the goals mentioned in the intro of finding challenges or identifying areas of SE where CV has been applied or not and the benefits (also implied in RQ4 on evaluation) and drawbacks of CV approaches. There seems to be value in papers investigating existing and well-known CV techniques even if no new approach for using that has been proposed. For example, industrial experience reports would be of value for your overall goals but not clear they would be found and included.

- Thank you for your comment. Our survey focused on primary publications as opposed to secondary studies. We analyse the pool of papers across novel dimensions and propose research questions that have not been investigated before. Therefore, by stipulating certain requirements on the paper's content (e.g. the nature of proposed approach), we are able to answer key research question details such as how a computer vision technique was used, what specific visual artifacts were used, and how were they analyzed. Answering such research questions and other details often requires the level of depth found in technical research publications. Nonetheless, we do include secondary industrial reports/case studies in our discussion section wherever relevant.

MA3. Scoping and search problems might have made relevant papers being missed.

- For example work by Coppola et al, Alegroth and Feldt and others have investigated the industrial application of visual GUI testing. Many of these papers investigate the rationales for introducing CV approaches as well as investigate, in detail, their drawbacks. It seems that these types of papers would have added value, at least to your RQ2. Thus it is not clear why they weren't found or why they were excluded. If if not included they might have helped in identifying different types of rationales in RQ2 so still seem relevant to your study. At least you need to clarify the scoping but potentially you need to extend your search/screening/extraction.

- Coppola, R., Morisio, M., and Torchiano, M. (2018a). Maintenance of android widget-based gui testing: A taxonomy of test case modification causes. In 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pages 151–158.

- Coppola, R., Morisio, M., Torchiano, M., and Ardito, L. (2019). Scripted gui testing of android open-source apps: evolution of test code and fragility causes. Empirical Software Engineering, pages 1–44.

- These are not visual (i.e. computer vision-based) approaches and therefore out of scope. The investigated testing tools are text-based; both element localization and test assertions are conducted via attributes and element IDs. Here is an excerpt from one of the papers:

> “all the considered GUI Testing Frameworks are [...] component-based GUI testing tool, that strongly focus on properties (e.g., ids, text, and positions in the layout hierarchies)”

Nonetheless, we will include some of them as related work in the revised manuscript.

- E. Alegroth, J. Gustafsson, H. Ivarsson, and R. Feldt, "Replicating Rare Software Failures with Visual GUI Testing: An Industrial Success Story", IEEE Software 34.5 (2017): pp 53-59.

- E. Alegroth, R. Feldt, and P. Kolstrom, "Maintenance of Automated Test Suites in Industry: An Empirical study on Visual GUI Testing", Information and Software Technology, vol. 73 (2016): pp. 66-80.

- E. Alegroth, R. Feldt and L. Ryrholm, "Visual GUI Testing in Practice: Challenges, Problems and Limitations", Empirical Software Engineering (2015), vol. 20: pp. 694-744. doi:10.1007/s10664-013-9293-5

- E. Börjesson and R. Feldt, "Automated System Testing using Visual GUI Testing Tools: A Comparative Study in Industry", 5th Int. Conf. on Software Testing, Verification and Validation (ICST), Montreal, Canada, April 17-19, 2012, pp. 350-359.

- We already cited a number of papers from the line of work by Alegroth/Börjesson et al.

- Another example is work by Chaudron et al on using Machine Learning to extract information from hand-drawn UML diagrams. This also requires some form of CV and it is not clear why you did not find/unclude such papers. [MH Osman, T Ho-Quang, M Chaudron: An automated approach for classifying reverse-engineered and forward-engineered UML class diagrams. 2018 44th Euromicro Conference]…

- Thank you for suggesting this reference. We might have missed it due to the venue (Euromicro) not being on our list of selected venues. We will add this to the pool.

Additional/minor comments:

- I don't really agree that the source code is the ultimate product of SE. IMHO the dynamically executing system is and this dynamic behavior may only quite indirectly be directed by the source code. At present, the link is often quite direct but in more autonomous and dynamically adapting systems it might not be. I thus question that the source code, per se, is quite as fundamental as you make it out in the introduction. Even something simple as code generation from some model might not be evidently included in what you view as "source code".

    - That’s a valid perspective, thanks for pointing that out. In any case, whether or not one considers source code as the ultimate product of SE, this does not impact the scope of the manuscript, which is to survey the use of computer vision in software engineering.

- Your focus on "SE research" perspective/approach/viewpoint is also a bit unclear to me. It is not clear if applications of CV to a specific software task is included in SE research or not. I assume you have a broad outlook here but since you talk both about SW development and about SE research and stress the latter in the introduction it leaves the reader unclear on what you really mean here. Is the scope somehow limited by your use of this phrase and if so how? If you include any use of CV in SE, even if reported for example in industry papers or experience reports, then why stress the notion of "research" early on? Think through this and clarify early on in the paper. For example you say that the CV perspective has "the final objective of addressing a SE research challenge or objective". Is it not enough that it is a SW dev challenge? Maybe your point is that you exclude grey literature and focus only on research papers but then I think it would be better to just state that early on and then drop the "research" part when discussing this.

    - Thank you for your comment. We use the term “research” to indicate that our focus is examining the software engineering research literature, as opposed to, for instance, technical magazine articles and other grey literature. This enables creating a pool of papers that have sufficiently detailed algorithms and evaluations, in order to answer key research questions related to the details of the visual approach, visual artifacts, and evaluation strategies.

- Not clear why you make a difference between visual artifact and visual approach (in 2.1).

    - A visual artifact is the piece of data representing the input or output of a visual approach. In other words, the visual approach is algorithm itself, and the visual artifact is the data fed to the algorithm; representing a visual aspect of software. We do describe this in 2.1, but we will clarify it further.

- Scope declaration talks about a "dispersed literature", but not clear in what sense this is so. It is not "a given" that your perspective of focusing on CV is a valuable one (for example if different SE tasks have very different "visual" characteristics) and without this perspective one cannot even talk about "a field" and thus cannot expect a unified literature. I think you should be more clear early on that this is a new perspective and you need to argue more for its worth (rather than assuming it). Note that I'm not saying the worth is not there; I'm making a point about your "stance" when reporting and arguing for the worth of your work.

    - Thank you for your comment. We say “dispersed literature” to describe that there exists a body of work that uses computer vision to address SE problems, but this body of work is dispersed and has not been visible enough to be considered by the research community as a viable paradigm to advance SE research.

- Criteria are not performed: "performed a number of quality assessment criteria".

- 38+8 is not equal to 42 so please describe the screening that went on in the snowballing step.

    - Thank you for pointing this out. This is a typo from a previous iteration of the manuscript and will be corrected in the revised version. Thank you

Reviewer #3

Overall I like the article as it contains several interesting insights. To me these are especially the list of papers (Table 3), the publication count statistics (Figures 3--5), and the holes in the existing work (Figures 8 & 9 and Table 6).

To me the major limitation of this article is that it does not do a good job of putting the survey into the context of prior surveys. The most closely related work [71] is only discussed in one paragraph on the last page. This treatment of a related survey is too little too late, as both surveys overlap significantly. To me this article should start by summarizing the findings of the earlier survey [71] and then formulate its research questions in relation to the earlier work. So what questions did [71] not answer, which questions need additional evidence, etc.

- Thank you for your comment. Reference [71] has a significantly different objective. It does not survey computer vision or visual techniques. Its objective and ours has no overlap. Their survey focuses specifically on the problem of cross-browser testing, regardless of the approach used. Our survey focuses on the use of computer vision algorithms in software engineering, regardless of the problem area (e.g. testing, requirements, maintenance or any other area). In other words, their focus is on surveying different categories of techniques to solve a specific browser testing problem, while our focus is on surveying how a certain category of techniques were utilized in all areas of software engineering. It is only remotely related to our work in the sense that they discovered that visual approaches happen to be one of the potential ways of cross-browser testing, and then they continue their survey by discussing other, non-visual, techniques (e.g. DOM analysis, navigation model analysis). Nonetheless, we will clarify this in the related work section in order to better contrast these two works.

The abstract states the article aims to "[..] provide a comprehensive survey of the current state of the art of the use of computer vision techniques in SE [..]". It is then a bit strange that the article completely ignores related work that is not found in published papers, i.e., work happening in industry such as in startup companies. Are these practical solutions not "state of the art"? While it may be hard to compare academic work with industry work, it would at least be interesting to see in which areas there are already commercial solutions. Some of this industrial work is relatively well publicized with blog posts or public APIs that are free to use (https://www.youtube.com/watch?v=rD868JYIc60) (https://teleporthq.io/blog/new-vision-api/).

- Thank you for your comment. Including commercial work would negatively impact our ability to conduct the survey. This is because commercial work does not include a detailed explanation of their approach. This reduces our ability to answer research questions (e.g. what specific computer vision techniques were used, what aspect of the visual artifact is used). Commercial work also does not include evaluation, and therefore we won't be able to answer research questions related to how computer vision techniques were evaluated, what were the main findings, and what are some of the challenges. We clarify this further in the revised version.

On the academic side, since the abstract describes the survey as comprehensive, it is not good that the article ignores several papers that should be included (or explained why they are not included), including the following:

Anabela Caetano, Neri Goulart, Manuel Fonseca, and Joaquim Jorge. 2002. JavaSketchIt: Issues in sketching the look of user interfaces. In Proc. AAAI Spring Symposium on Sketch Understanding. AAAI, 9–14.

Adrien Coyette, Suzanne Kieffer, and Jean Vanderdonckt. 2007. Multi-fidelity prototyping of user interfaces. In Proc. 11th IFIP TC 13 International Conference on Human-Computer Interaction (INTERACT). Springer, 150–164.

Morgan Dixon and James Fogarty. 2010. Prefab: Implementing advanced behaviors using pixel-based reverse engineering of interface structure. In Proc. ACM SIGCHI Conference on Human Factors in Computing Systems (CHI). ACM, 1525–1534.

Morgan Dixon, Daniel Leventhal, and James Fogarty. 2011. Content and hierarchy in pixel-based methods for reverse engineering interface structure. In Proc. ACM SIGCHI Conference on Human Factors in Computing Systems (CHI). ACM, 969–978.

James A. Landay and Brad A. Myers. 2001. Sketching interfaces: Toward more human interface design. IEEE Computer 34, 3 (March 2001), 56–64.

Julian Seifert, Bastian Pfleging, Elba del Carmen Valderrama Bahamóndez, Martin Hermes, Enrico Rukzio, and Albrecht Schmidt. 2011. MobiDev: A tool for creating apps on mobile phones. In Proc. 13th Conference on Human-Computer Interaction with Mobile Devices and Services (Mobile HCI). ACM, 109–112.

Siva Natarajan and Christoph Csallner. 2018. P2A: A tool for converting pixels to animated mobile application user interfaces. In Proc. 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft). 224-235.

Forrest Huang, John F. Canny, and Jeffrey Nichols. 2019. Swire: Sketch-based User Interface Retrieval. In Proc. CHI Conference on Human Factors in Computing Systems (CHI). ACM.

- Thank you for drawing our attention to these papers. They were not included for the following reasons. 1) Some of them were published after the survey was conducted. 2) They were not detected earlier because the venues are not among software engineering-specific venues we used, and they were not retrieved by the database search. We will include these papers in our manuscript. However, we note that the papers do not add new categories or introduce classification change. They are simply more examples of categories and classifications that we have already defined in the survey.

I would reword RQ1 from "What are the main software engineering areas and tasks for which computer vision approaches are used?" to "What are the main software engineering areas and tasks for which computer vision approaches have been used to date?". Otherwise the article may get outdated quickly.

- Thank you for the suggestion. We will take that into consideration.

Figure 1 is quite confusing:

-- Why does the figure contain these particular software artifacts?

-- Why are the visual artifacts not part of the software artifacts? Definition 1 seems to indicate visual artifacts are indeed software artifacts.

-- What is this processing pipeline?

- Thank you for your comment. We will clarify the labeling and caption of the figure, as well as elaborate on its description in the text. The software artifacts in the figure are some examples of common artifacts used in various stages of software engineering (e.g. development artifacts, testing artifacts, maintenance artifacts). Visual artifacts are then constructed from these artifacts. The visual processing pipeline simply indicates that the existing software artifacts are converted/processed into visual artifacts and then analyzed using computer vision, in order to finally achieve a software engineering task. The figure and corresponding text will be clarified in the revised version.

Figure 2 first applies the "inclusion criteria" and then the "exclusion criteria". But according to the text, the inclusion criteria already handle the area mismatch (i.e., not software engineering) exclusions the paper assigns to the "exclusion criteria". This needs clarification.

- The differences between inclusion and exclusion criteria can be a gray area. That is, what might be an exclusion criterion can be casted into an equivalent inclusion criteria and vice versa. In our context, inclusion criteria are meant to cast a wider net and collect as many relevant papers as possible. However, because these criteria tend to be broad (by design), they often need further inspection to remove irrelevant work. Inclusion criteria are therefore applied in a broad coarse-grained way, while exclusion criteria are more fine grained. For instance, in our inclusion criteria, we check that a given paper appears to be using a visual technique and is related to software engineering. However, in exclusion criteria, a fine-grained and more detailed inspection of the work indicates that it is actually a visualization approach, rather than a computer vision approach. We will add this clarification to the revised text.

Figure 2 and the text both mention that the process has a step of applying "quality criteria". But the article never explains or even lists these quality criteria. Is there a subsection missing to explain these quality criteria (between Sections 2.6 and 2.7)?

- Thank you for pointing this out. This is a typo from a previous iteration of the manuscript and will be corrected in the revised version. Thank you.

Since the article already distinguishes between conference and journal papers, it would be interesting to compare these communication channels based on the studied papers. Do papers selected for this survey pool cite other papers in the pool? If yes, do they cite more conference or journal papers? Maybe we could normalize this comparison by publication year.

Related to the above, why does Table 2 not include citation counts (e.g., from Google Scholar)?

- Thank you for the suggestion. We will take it into consideration.

Figure 4 should be a step-function. Currently it looks like there was a point in time where 1.23456236 papers on testing were published. Also why are there minor lines at 2.5 papers published, 7.5, etc.? Figure 4 should also use line types that are easier to distinguish.

- The figure is a cumulative plot of publications across years. We used a line plot to clarify the trend of each SE area, since a plot of only dots was confusing and made it hard to notice any trend. The minor lines are only a visual aid. We also used all line types possible (solid, dotted, dashed, with different spacings) to make the figure easier to understand.

Why does Figure 6 show 6 papers published for "automated code generation", but the text only cites 5?

- Thank you for pointing this out. This is a typo from a previous iteration of the manuscript and will be corrected in the revised version. Thank you.

Table 5 seems to be missing a row on "machine learning", as the text has a separate paragraph on it (just like for the other techniques). Table 5 also seems a bit strange as it mixes elementary computer vision operations with applications that combine several of such operations (e.g., OCR).

- Thanks for the comment. 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. As for your question on adding a machine learning row, we would like to point out that machine learning does not really fit in table 5 since it is not a specific computer vision algorithm. Rather, it is complementary and independent of the particular computer vision used, as it can be equally used in image diff or any of the other algorithms listed in the table. Therefore, it does not really fit in table 5.

Section 3.4.4 stands out in that it does not back up its finding with numbers. It should be easy to summarize how many papers used each of the named libraries.

- Thanks for pointing this out. We will add the number of papers to the section.

Section 4 could be strengthened by mentioning what overlap exists between the papers' evaluation benchmarks. The paper hints at little overlap, but it would still be valuable to mention any benchmark that may have been used in more than one paper.

Why does Section 4 claim that no benchmarks exist? What about Rico?

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In Proc. 30th Annual ACM Symposium on User Interface Software and Technology, pp. 845–854. ACM, 2017.

- Thank you for the question. The paper you mentioned, Rico, is already in our collected pool of papers (see Table 1). It is the row that starts with “Deka et al.”.

  Section 4 (Discussion) points out the lack of categorized and labeled repository of visual artifacts that might serve evaluation benchmarks. While we do point out that such benchmarks exist for non-visual software engineering research and give a few examples, this has not been the case for techniques using computer vision. The paper/dataset you mentioned, Rico, is not a benchmark. It is an unlabeled dataset collected from recordings of user interactions. No labeling or ground truth information is provided. The authors do not claim or propose that it is a benchmark.

The article contains several claims that need evidence, citations, or toning down. Following are examples:

-- Section 3.1 claims to determine which SE area has "benefited the most from visual approaches". Does this assume that "more publications" implies "more benefit"? Playing devil's advocate here, a publication may actually hurt a research area by publishing bogus results. So this "benefit" needs further explanation.

- Thank you for pointing this out. We will rephrase the sentence to better reflect the scope and research questions.

-- Why does Section 3.5.1 claim that it is "unfortunate" that several papers do not include an industrial evaluation? Of course such evaluations can add value to certain papers, but other papers (e.g., on early iterations of a technique) may be perfectly valuable without including such evaluations. In other words, should we really delay publishing all techniques until they are developed to a point where it can withhold industrial evaluation?

- That’s a valid perspective, thank you for pointing it out. We will rephrase the sentence to have a better wording.

-- Section 4 claims that "most of software developed nowadays has a GUI or other visual interfaces" but does not back up this claim. How would you even measure this, given that much software development is proprietary and only used inside a single organization?

- It would be practically impossible to measure the number of all software that uses a GUI as a ratio of all software. We are not aware of any work claiming to estimate this number either. The sentence was simply meant to point out that software using GUI is quite common. We will remove or rephrase the sentence.

-- The claim about increasing number of software engineering papers using computer vision should be normalized with the total number of software engineering papers published (which may also increase over the same time span).

- Thank you for the suggestion. First, this may not be a viable evidence. If the numbers were normalized, an upward or downward trend might be due to an increase/decrease in other areas of software engineering, not the increase of computer vision papers. Second, figuring out the total number of software engineering papers is not straightforward and can not be measured unambiguously. Accordingly, what we mean by the sentence you referred to is simply that the number of papers using computer vision is increasing.

RQ4: "How are software engineering tasks which leverage" --> "How are software engineering tasks that leverage"

- Thanks, will rephrase.

Page 6 should define "SRL" before using it. Also it is a typo (SLR), right?

- That is correct, it is a typo. Thanks for pointing this out.

Figures 3 and 5 should not start their y-axis at -0.5 papers published. Also, why are there minor lines at 0.5, 1.5, 2.5, etc. papers published?

- Thanks, will fix accordingly.

The last paragraph of Section 3.3.2 seems out of place (move to different subsection?) as it talks about the higher precision of visual techniques, while the section is titled "ease of use".

- It is actually a continuation of the preceding paragraphs. But you are correct, it does break the flow. We will rephrase to better link with the rest of the section.

Section 3.3.3 seems to mix up references [41] and [42].

- Thanks for the comment. These are actually two separate references and we don’t follow in what way are they being mixed up.

Section 3.4.1 introduces categories that may better be named differently:

-- "temporal artifact" --> "video artifact"

-- "natural input" --> "sensor input"

-- "localized" --> "partial"

- Thanks for the suggestions. We believe the existing labels better reflect our synthesis of the findings. For instance, not all natural inputs are sensor inputs. Some approaches capture natural inputs through sensors (e.g. tablets), while others capture the same natural inputs using mouse gestures or photographs.

Why does the second paragraph of Section 3.4.1 only mention testing?

- We mention testing by way of example. We do not state that the artifact is exclusive to testing. However, testing does represent the vast majority of publications in the collected pool of papers and we state that in our findings.

The wording in Section 3.5.2 on TV ads is strange as it seems to imply they only became annoying a year ago. The same paragraph also only focuses on web pages, but it is not clear why.

- It is citing a finding that states that the number and views of web ads have recently overtaken TV ads. The focus in that sentence is on web pages because, when conducting visual analysis, the ads can interfere with the correct functioning of the algorithm and decrease its robustness.