Computer Standards & Interfaces 92 (2025) 103925

Contents lists available at ScienceDirect

Computer Standards & Interfaces

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

Delta GUI change detection using inferred models

Fernando Pastor Ricós [a,*], Beatriz Marín [a], Tanja E.J. Vos [a,b], Rick Neeft [b], Pekka Aho [b]

[a] Universitat Politècnica de València, València, Spain
[b] Open Universiteit, The Netherlands

ARTICLE INFO

Keywords:
Delta validation
GUI change detection
Scriptless testing
State model inference
Systematic mapping

ABSTRACT

Recent software development methodologies emphasize iterative and incremental evolution to align with stakeholders' needs. This perpetual and rapid software evolution demands ongoing research into verification practices and technologies that ensure swift responsiveness and effective management of software delta increments. Strategies such as code review have been widely adopted for development and verification, ensuring readability and consistency in the delta increments of software projects. However, the integration of techniques to detect and visually report delta changes within the Graphical User Interface (GUI) software applications remains an underutilized process. In this paper, we set out to achieve two objectives. First, we aim to conduct a comprehensive review of existing studies concerning GUI change detection in desktop, web, and mobile applications to recognize common practices. Second, we introduce a novel change detection tool capable of highlighting delta GUI changes for this diverse range of applications. To accomplish our first objective, we performed a systematic mapping of the literature using the Scopus database. To address the second objective, we designed and developed a GUI change detection tool. This tool simultaneously transits and compares state models inferred by a scriptless testing tool, enabling the detection and highlighting of GUI changes to detect the widgets or functionalities that have been added, removed, or modified. Our study reveals the existence of a multitude of techniques for change detection in specific GUI systems with different objectives. However, there is no widely adopted technique suitable for the diverse range of existing desktop, web, and mobile applications. Our tool and findings demonstrate the effectiveness of using inferred state models to highlight between 8 and 20 GUI changes in software delta increments containing a large number of changes over months and between 4 and 6 GUI changes in delta increments of small iterations performed over multiple weeks. Moreover, some of these changes were recognized by the software developers as GUI failures that required a fix. Finally, we expose the motivation for using this technique to help developers and testers analyze GUI changes to validate delta increments and detect potential GUI failures, thereby fostering knowledge dissemination and paving the way to standard practices.

1. Introduction

In recent decades, iterative and incremental software development methodologies have gained widespread acceptance in a diversity of projects ranging from small and large industries to personal and research projects. The broad adoption of these methodologies, such as agile practices, arises from their pivotal role in accelerating the dynamic and rapid evolution of software [1–3]. As the needs and expectations of software stakeholders continue to evolve, there is an increased emphasis on continuous development and rigorous verification as fundamental tasks. Consequently, it becomes imperative to embrace practices, values, and principles that facilitate swift responsiveness to software changes, whether they stem from the environment, user requirements, or delivery constraints [4].

In rapidly iterative and incremental software development projects, there is a high emphasis on facilitating the seamless integration of developers' changes into shared software repositories. These integrated repositories serve as a foundation for building every commit, running tests, documenting requirements, and evaluating the overall quality of the source code [5].

The concept of *delta change* within software development projects generally refers to any modification or update from one software version to the subsequent one [6]. This incremental change, known as delta increment, can encompass a wide range of modifications, regardless of their scale. For instance, it might involve preparing a new software delta version, wherein multiple delta increments occur, such as the development of new functionalities, bug resolution, integration of unit tests, or updating documentation.

* Corresponding author.
E-mail address: fpastor@pros.upv.es (F. Pastor Ricós).

https://doi.org/10.1016/j.csi.2024.103925
Received 28 March 2024; Received in revised form 3 July 2024; Accepted 2 September 2024
Available online 3 September 2024

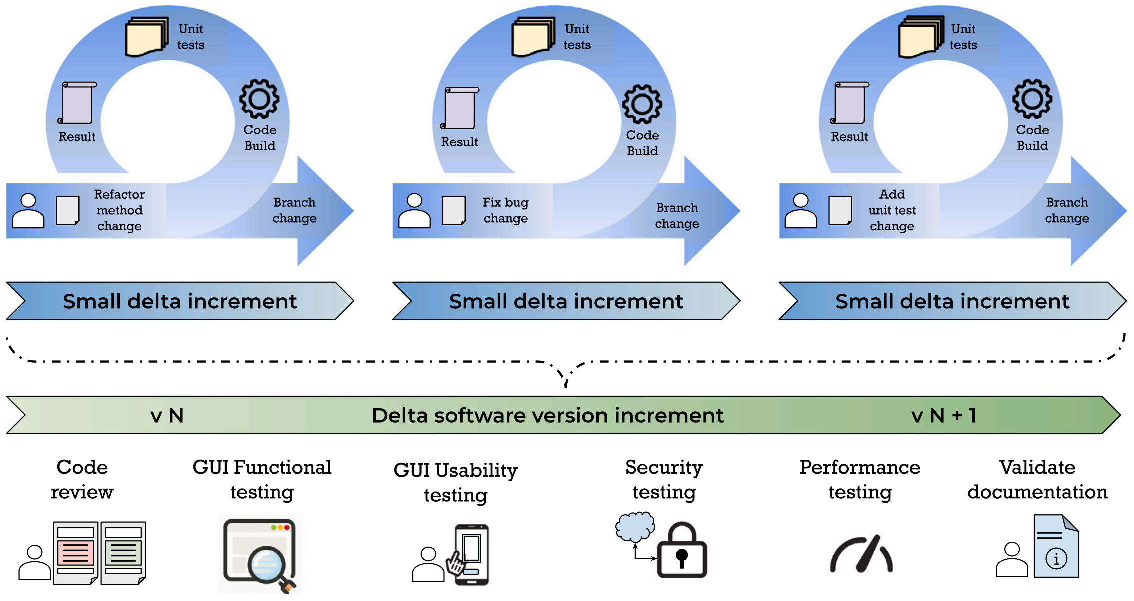


Fig. 1. Overview of a delta testing process in software projects.

In software projects characterized by frequent delta increments that bring changes in the software product, it is of paramount importance to implement testing processes that streamline the analysis and verification of these increments. The diverse delta developments, whether introducing, removing, updating, or refactoring internal software functionalities, may unintentionally introduce failures not only in the implemented methods but also in the dependencies of classes and libraries. The testing process should focus on detecting, controlling, and minimizing these potential failures. Moreover, while human testers' expertise and tacit knowledge are essential for testing and analyzing results [7], human time is limited, especially in rapidly evolving projects. Therefore, it is necessary to implement tools that automate and complement the human efforts [8].

Depending on the size of the delta increment, the testing process may require different types of analyses and verification techniques (see Fig. 1). Minor delta increments, which involve small changes in code methods, may be adequately validated through software compilation and unit testing. On the other hand, larger increments, such as new delta versions of a software product, demand additional analysis and verification by multiple collaborators and techniques on the project, including usability, functional, and security testing.

As code undergoes continuous evolution via delta increments, specific techniques, such as code reviews, have emerged as critical strategies for development and verification, ensuring readability and consistency in software projects [9]. This review process not only enhances code quality but also stands as one of the most effective means to uncover bugs and drive continuous improvements within the codebase [10].

Recognizing delta changes within software code and performing code peer reviews has become standard practice in software projects [11]. However, the analysis and validation of delta changes within the Graphical User Interface (GUI) remains an underutilized process across desktop, web, and mobile applications. This oversight is unfortunate, given that the GUI serves as the primary point of contact between users and the software system, making it a crucial component for ensuring usability, functionality, security, performance, and more. Overlooking GUI delta changes neglects a fundamental aspect of software quality, potentially impacting the user experience and the overall effectiveness of the application.

We consider that integrating delta change detection techniques to visualize, analyze, and review GUI delta changes stands as a significant step towards comprehensive software quality assurance. By automating this approach, developers, testers, and other project contributors can streamline their efforts in identifying functionalities that have been removed, added, or modified. The information provided by the automated technique serves as a validation step, ensuring that the implemented code adequately reflects the intended GUI changes or uncovers unexpected GUI modifications that can be considered bugs.

For a deeper understanding of the *GUI change detection* practices, we have performed a systematic mapping of the literature to shed light on the current state of knowledge in the field. After that, we developed a novel tool to detect and highlight delta GUI changes of one desktop, web, and Android open-source applications. Therefore, the main contributions of this paper are:

- A systematic mapping of the literature that studies the fundamental concepts, methods, and technologies associated with GUI change detection.
- A GUI change detection tool that compares and highlights the GUI changes between different versions of desktop, web, and Android GUI systems.

The significance of these contributions extends to both academic and industrial practitioners. Firstly, the systematic mapping of the literature collects and provides insights into the state-of-the-art of delta GUI change detection techniques. Secondly, the designed GUI change detection tool can motivate practitioners to integrate this technique into their software projects or encourage them to enhance this innovative technique with further research.

This paper is structured as follows. Section 2 presents the systematic mapping of the literature of the field of GUI change detection. Section 3 describes the GUI State Model inference process. Section 4 presents the GUI change detection tool. Section 5 details the empirical study to evaluate the GUI change detection approach. Section 6 shows the results obtained. Section 7 describes the threats to the validity. Section 8 presents a discussion that analyzes the empirical findings and suggests further research directions. Finally, Section 9 exposes our conclusions and future work.

2. Literature review

In [12], we describe a first proof-of-concept of our *Delta GUI Change Detection* approach. However, the continuous evolution of concepts and

```
TITLE-ABS-KEY(((gui OR ui OR "graphical user interface" OR "user interface") W/2 chang*) AND detect*)
OR TITLE-ABS-KEY((detect* W/2 (gui OR ui OR "graphical user interface" OR "user interface")) AND chang*)
OR TITLE-ABS-KEY((chang* W/2 detect*) AND (gui OR ui OR "graphical user interface" OR "user interface"))

OR TITLE-ABS-KEY(((gui OR ui OR "graphical user interface" OR "user interface") W/2 chang*) AND delta)
OR TITLE-ABS-KEY((delta W/2 (gui OR ui OR "graphical user interface" OR "user interface")) AND chang*)
OR TITLE-ABS-KEY((chang* W/2 delta ) AND (gui OR ui OR "graphical user interface" OR "user interface"))

OR TITLE-ABS-KEY(((gui OR ui OR "graphical user interface" OR "user interface") W/2 chang*) AND (evolv* OR evolut*))
OR TITLE-ABS-KEY(((evolv* OR evolut*) W/2 (gui OR ui OR "graphical user interface" OR "user interface")) AND chang*)
OR TITLE-ABS-KEY((chang* W/2 (evolv* OR evolut*)) AND (gui OR ui OR "graphical user interface" OR "user interface"))

OR TITLE-ABS-KEY(((gui OR ui OR "graphical user interface" OR "user interface") W/2 chang*) AND report*)
OR TITLE-ABS-KEY((report* W/2 (gui OR ui OR "graphical user interface" OR "user interface")) AND chang*)
OR TITLE-ABS-KEY((chang* W/2 report*) AND (gui OR ui OR "graphical user interface" OR "user interface"))

OR TITLE-ABS-KEY(((gui OR ui OR "graphical user interface" OR "user interface") W/2 differen*) AND detect*)
OR TITLE-ABS-KEY((detect* W/2 (gui OR ui OR "graphical user interface" OR "user interface")) AND differen*)
OR TITLE-ABS-KEY((differen* W/2 detect*) AND (gui OR ui OR "graphical user interface" OR "user interface"))

AND ( LIMIT-TO ( LANGUAGE , "English" ) OR LIMIT-TO ( LANGUAGE , "t PUBYEAR < 2023" ) )
AND ( LIMIT-TO ( SUBJAREA , "COMP" ) OR LIMIT-TO ( SUBJAREA , "ENGI" ) )
```

Fig. 2. Search query to embrace diverse terminology for GUI delta changes.

terminologies within GUI testing, demand for a comprehensive understanding of the existing research literature. To this end, we conducted a systematic mapping of the literature following the methodological guidelines of Kitchenham et al. [13,14]. In order to guide this study, we have formulated the following research question:

- RQ1: What techniques are employed for delta GUI change detection, and for which type of systems?

2.1. Methodology

We have chosen Scopus as our primary database choice. In comparison with other scientific repositories, such as Web of Science (WoS), Scopus offers broader and more inclusive content coverage [15], ensuring that we can access an extensive range of relevant research. Furthermore, Scopus provides robust impact indicators that are less susceptible to manipulation and are available for all serial sources in all disciplines [16].

Initially, we established the formulation of a search query to retrieve the studies related to the delta GUI change detection approach, which encompass the initial terms `delta`, `GUI`, `change`, and `detection`. However, based on insights gained from our previous research [12], we recognized the limitation of these terms as other studies employ different terminology to disseminate the use of GUI change detection techniques. For this reason, we decided to extend the query by incorporating the term `evolve` to capture descriptions of releasing changes as new application versions, `report` to signify the process of informing and presenting change results, and `difference` to address methods capable of identifying and detecting changes. Moreover, recognizing these terms can be combined with versatile interrelationships, we opted to use multiple `OR` disjunction combinations to ensure a comprehensive search. The final search query formula is shown in Fig. 2:

- The Scopus operator **TITLE-ABS-KEY** specifies that the terms in the query should appear in the title, abstract, or keywords of the scientific publication.
- The term `gui` is expanded to include `ui`, `graphical user interface`, and `user interface` in order to encompass all references to the system's user interface.
- The term `evolve` is expanded to include `evolution`.
- The wildcard character * is used for the terms `detect*`, `chang*`, `evolv*`, `evolut*`, `report*`, and `differen*`, allowing for variations in word endings and enhancing the search's inclusivity.
- This query generates multiple combinations of terms to explore versatile interrelationships between them, reducing the likelihood of missing relevant papers.
- The `W/2` operator sets the minimum distance between related terms to 2 words, ensuring a reasonably close relationship between them in the document.
- The query restricts results to publications in the `English` language.
- The scope focuses on publications in the subject areas of Computer Science (`COMP`) and Engineering (`ENGI`).

We decided not to limit the search to specific years in order to retrieve all existing studies. The Scopus search retrieved a total of 820 papers in the range from 1985 to September 2023. Then, with the aim to identify relevant research papers within the field of GUI software testing, more specifically about GUI change detection approaches, we established an iterative exclusion process that is depicted in Fig. 3 and explained below.

(1) **Exclusion Based on Field Relevance (Abstract):** In the first step of the exclusion criteria, we read the abstracts to remove papers not directly related to the GUI software testing field. This meant excluding papers that fell into other domains, such as topology, physics, structural engineering, or medical research. Furthermore, we excluded publications that are not conference, workshop, journal publications, or book chapters. After this initial exclusion, 132 papers remained out of the original 820.
(2) **Exclusion Based on GUI Change Detection (Abstract):** The second step involved refining the selection, focusing on papers abstracts related to the detection of GUI changes. This refinement process entailed excluding papers that primarily discussed topics such as identifying GUI state elements or analyzing the aesthetic aspects of GUIs without comparing version changes. As a result, 41 papers were retained from the subset of 132, ensuring a more targeted set of research papers aligned with the study's objectives.
(3) **Paper selection based on Report GUI delta changes:** In the third step, we conducted a comprehensive reading of the 41 papers. The focus was on identifying papers that specifically addressed the creation of textual or visual reports to inform about the GUI delta changes between different versions of the System Under Test (SUT). Furthermore, those studies must contain an empirical evaluation that experiments and validates the proposed technique. Papers that exemplify the technique description but do not indicate any evaluation are discarded. As a result, 6 papers, without considering our previous work, were finally selected from the 41 GUI change detection papers of the original group of 820 queried papers.
As an example of an excluded paper, the study by Bures [17] proposes an approach designed to track delta UI changes in software development projects, providing valuable information for test designers. For instance, this information can be employed in the maintenance of test scripts. However, the way in which this information is reported to stakeholders remains unspecified. Furthermore, the simulations or experiments mentioned appear

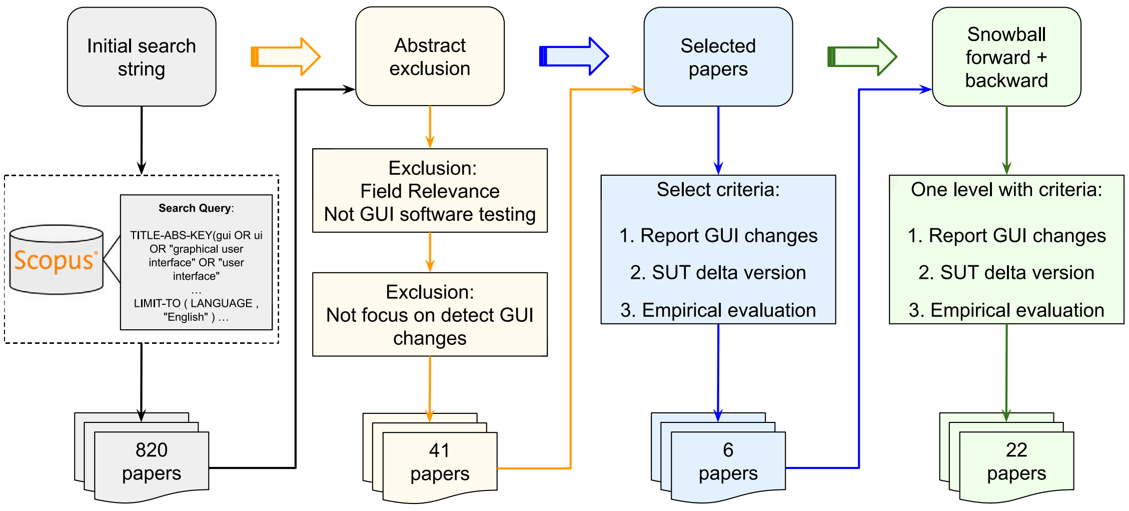


Fig. 3. Overview of the process for the systematic mapping literature review.

more as a prospective estimation of the proposal rather than an empirical evaluation.

(4) **Snowball paper selection on Report GUI delta changes:** The final step consists of applying one level of backward and forward snowballing process [18]. Since we consider that relevant GUI change detection techniques can be related to the subset of 41 papers, we applied the snowball process in all this subset instead of only the 6 selected ones. The snowball inclusion criteria remain the same as the third step: papers that focus on the creation of textual or visual reports to inform about the GUI delta changes between different versions of the SUT, and that must contain some empirical evaluation that experiments and validates the proposed technique. This snowballing process included 16 additional research papers, which made up a total of 22 GUI change detection papers.
In this snowball process, we found GUI change detection research that does not specifically mention or exemplify how changes are reported to users. As an example, the study by Grechanik et al. [19] presents a tool called GUIDE that allows users to differentiate the GUIs of evolving GUI-based application versions. Nonetheless, because the study seems focused on presenting the algorithm improvement, there is a lack of information that indicates how the differencing results are reported to the users.

2.2. Data collection

For each selected paper, we read the whole paper and extracted the following information in an Excel file:

- **Type of SUT**: Identification of the SUT type for which the technique is implemented (desktop, web, mobile, image-based).
- **Change Detection Technique/Algorithm**: Summary of the technique or algorithm employed for GUI change detection (Web Document Object Model (DOM) tree or XPath comparison, Hash calculations and comparison, Visual recognition and comparison techniques, etc.).
- **GUI Data Extraction**: Description of how the GUI information is extracted or utilized to apply the change detection technique or algorithm (Web URL, HTML document, XML data, State screenshot, etc.).
- **Type of Report**: Indication about how the detected changes are reported and/or highlighted to the user.
- **Empirical Evaluation**: Insight into how the evaluation of the technique or algorithm was conducted.
- **Open Source Availability**: Indication of whether the technique or algorithm is available as open source. Nonetheless, this is complementary and not considered for the exclusion criteria.

2.3. Data results

The predominant area of research in detecting delta GUI changes, accounting for 59% (13 out of 22) of total papers, is aligned with Change Detection and Notification (**CDN**) systems or tools for monitoring web applications. These systems initiate a crawling process on a set of web pages specified by the user. Subsequently, a comparison method or algorithm is applied to identify the values of diverse properties associated with Document Object Model (DOM) elements that have changed. Alternatively, these systems may use visual comparison techniques to compare the web states for detecting changes. Finally, the identified changes are notified to users and exposed through reports that incorporate textual and visual information.

Research on **regression** testing, with a focus on automatic detection and reporting of delta GUI changes, accounts for 32% (7 out of 22) of total papers and encompasses diverse approaches. First, web regression testing studies from Raina et al. [20] and Walsh et al. [21] employ similar DOM comparison techniques related to CDN systems. Second, regression testing studies that delve into the automatic inference of event graphs for web systems (Roest et al. [22]) or desktop systems (Gao et al. [23]) emphasize the reuse of event graphs for different system versions to identify state changes. Third, another research strand consisting of multiple studies (Tanno et al. [24]; Adachi et al. [25]) introduces image-based methods, potentially integrable with other GUI testing tools that compare screenshots of application states between two versions to detect differences. Fourth, a study focusing on Android systems (Xiong et al. [26]) employs random GUI testing on an application version, repeating the same actions on the second version to detect widget inconsistencies.

Lastly, **other** papers account for 9% (2 out of 22) and describe tools with the capability of reporting GUI delta changes. TAO [27] is a GUI testing toolset that, in addition to including an automatic test generator and static binary analysis, includes a UI-Diff tool to track GUI changes on desktop applications. GCAT [28] is a tool with the objective of detecting and summarizing delta GUI changes during the evolution of mobile apps.

The scientific publications resulting from the systematic mapping of the literature are shown in Table 1.

Table 1
GUI Change Detection research papers obtained from the systematic mapping of the literature.

Year	Title	SUT	Group	Summary
2001	Monitoring Web information changes [29]	Web	CDN	A system called CDWeb that allows users to monitor a whole web document or specific portions of their interest.
2001	WebSCAN: Discovering and Notifying Important Changes of Web Sites [30]	Web	CDN	A system called WebSCAN that monitors and analyzes the change of pre-registered Web sites and notifies important changes to users.
2001	Perception of content, structure, and presentation changes in Web-based hypertext [31]	Web	CDN	A study about how users perceive web changes and which changes they consider relevant. Then, the observations of the study were used to guide the design and development of Walden's Paths Path Manager tool.
2001	Managing change on the web [32]	Web	CDN	A detailed description of the Walden's Paths Path Manager tool used to assist maintainers in discovering when relevant changes occur to linked web resources.
2002	Information Monitoring on the Web: A Scalable Solution [33]	Web	CDN	A tool called WebCQ designed to discover and detect changes on web pages and notify users about interesting changes with personalized messages.
2004	The eShopmonitor: A comprehensive data extraction tool for monitoring Web sites [34]	Web	CDN	The eShopmonitor is a tool that allows users to monitor data of interest that has changed on commercial websites.
2004	Managing distributed collections: Evaluating Web page changes, movement, and replacement [35]	Web	CDN	The Walden's Paths Path Manager is a tool that allows users to monitor whether any page in a collection of web pages has changed.
2005	CX-DIFF: a change detection algorithm for XML content and change visualization for WebVigiL [36]	Web	CDN	A system called WebVigil that allows users to specify, manage, receive notifications, and view customized web page changes.
2009	TAO project: An intuitive application UI test toolset [27]	Desktop	Other	A GUI testing toolset called TAO that contains a UI-diff tool that allows to automatically track GUI changes.
2009	Changing how people view changes on the web [37]	Web	CDN	An Internet Explorer browser plugin that compares the previous cached page with the current page to highlight the way a page has changed when the user returns to it.
2009	Browsing Assistant for Changing Pages [38]	Web	CDN	A framework that provides continuous assistance to users browsing the Web regarding its temporal context.
2010	A novel approach for web page change detection system [39]	Web	CDN	A system that compares old and modified web pages to find and highlight the changes to the users.
2010	Vi-DIFF: Understanding Web Pages Changes [40]	Web	CDN	An approach called Vi-DIFF that detects content and structural changes in the visual representation of web pages.
2010	Regression Testing Ajax Applications: Coping with Dynamism [22]	Web	Regression	Crawljax is a tool that can automatically infer web state-flow graphs. Re-using information from previous web version graphs allows performing regression testing to view the added and removed states in each crawl session.
2013	An automated tool for regression testing in web applications [20]	Web	Regression	An automated tool for regression testing that can identify and report the changes in web applications.
2015	Pushing the limits on automation in GUI regression testing [23]	Desktop	Regression	GUITAR is a tool that can automatically infer event flow graphs. Re-using information from previous SUT version graphs allows performing regression testing to report widget and state mismatches.
2018	Detecting and summarizing GUI changes in evolving mobile apps [28]	Mobile	Other	A tool called GCAT for detecting and summarizing GUI changes during the evolution of mobile apps.
2020	Region-based detection of essential differences in image-based visual regression testing [24]	Image based	Regression	A visual regression testing method called ReBDiff that detects differences in the state images of two versions of an application.
2020	A Method to Mask Dynamic Content Areas Based on Positional Relationship of Screen Elements for Visual Regression Testing [25]	Image based	Regression	A visual regression testing method that allows the masking of dynamic state content when detecting differences in the state images of two versions of an application.
2020	Automatically identifying potential regressions in the layout of responsive web pages [21]	Web	Regression	A tool called REDECHECK that extracts the responsive layout of two versions of a web page and compares them, alerting developers to the differences in layout that they may wish to investigate further.
2021	WebEvo: taming web application evolution via detecting semantic structure changes [41]	Web	CDN	A tool called WebEvo for monitoring web element changes considering text and image content.
2023	An empirical study of functional bugs in Android apps [26]	Mobile	Regression	A tool called RegDroid that generates random GUI tests on two app versions and checks whether the GUI states of versions A and B contain similar widgets. If not, the inconsistency is reported as a bug.

2.4. *Other interesting related work topics*

Although we made the decision to exclude certain studies due to their lack of specific focus on reporting textual or visual GUI changes in delta versions, it is relevant to acknowledge and discuss research that employs GUI change detection techniques for diverse and important objectives within software testing:

2.4.1. *Repair test scripts*

Within the realm of research dedicated to detecting GUI changes to automatically repair or provide information to repair scripts, certain studies offer direct relevance. Determining its inclusion as directly or indirectly related work is a nuanced distinction.

Grechanik et al. [42] research combines a GUI diff-tree tool with a script analyzer tool. This combination aims to generate informative messages for test engineers, aiding them in the maintenance and evolution of GUI-directed test scripts that may break when GUIs are modified between successive releases of GUI-based applications.

Zhang et al. [43] introduce FlowFixer, a tool for Java Swing desktop applications. FlowFixer instruments the old application version by recording user actions and instruments the new application version by executing random UI actions. The objective is to identify instrumented method matches and, consequently, automatically repair broken workflows in evolving GUI applications. Gao et al. [44] evaluate the SITAR technique with Java desktop applications. SITAR utilizes GUI ripping to obtain an event flow graph that represents the GUI event interactions of a new application version. In cases where a test script cannot complete a path of events for execution, SITAR leverages information from the event flow graph to calculate multiple alternative repairing paths. Human testers are then notified about these alternatives to manually confirm how to repair broken scripts.

In the realm of web systems, several studies delve into strategies for repairing test scripts between two versions of websites. Choudhary et al. [45] introduce the tool WATER, which compares the behavior of test cases across successive releases of a web application to suggest repairs for broken tests. Hammoudi et al. [46] present WATERFALL, an approach that enhances the effectiveness of WATER by employing a fine-grained approach applied to the iterative versions/commits of web applications. Stocco et al. [47] propose a test repair technique that employs visual analysis implemented in a tool named VISTA. Kirinuki et al. [48] introduce the COLOR approach, which utilizes various web properties to support repairing broken locators in test scripts. Nass et al. [49] present the Similo approach, leveraging information from multiple web element locator parameters to identify the web element with the highest similarity.

In the domain of mobile systems, various studies explore methods to repair test scripts. Li et al. [50] introduce the ATOM approach, incorporating semi-automatic mechanisms to calculate delta event sequence models. These models capture changes introduced by new application versions, facilitating the maintenance of GUI test scripts. Song et al. [51] present an XPath-based approach that enables the repair and reuse of test scripts for subsequent app versions, particularly when alterations occur in the locations, names, or property values of UI controls. Chang et al. [52] propose the CHATEM approach, which involves the semi-automatic construction of event sequence models of evolving mobile apps to maintain and generate new test scripts. Pan et al. [53] develop the METER approach, leveraging computer vision techniques to execute test script actions in the next application version. This process allows the construction of the replacement of test actions to repair broken scripts. Extending the METER approach, Xu et al. [54] present GUIDER, which incorporates structural information of app GUIs to enhance the effectiveness of repairing test scripts.

2.4.2. *Cross-browser and cross-device testing*

While studies focused on testing GUI cross-browser compatibility for web applications or GUI cross-device compatibility for mobile systems do not directly employ GUI change detection for delta versions, we find it pertinent to mention certain tools and techniques that offer valuable insights for their potential usage.

Mesbah et al. [55] introduce an automated cross-browser compatibility testing approach in a tool called CrossT, an extension of the web crawler Crawljax. Initially, the web application is crawled in each desired browser to infer a navigation model. Subsequently, a pair-wise comparison is performed on the generated models to report discrepancies. Tanaka et al. [56,57] present the web application compatibility testing tool X-Brot, encompassing both functional and visual compatibility testing. The functional approach examines whether the same UI action on a web page results in a compatible action on each browser. Simultaneously, the visual technique checks whether the same web page displays a similar view on each browser. Ren et al. [58] propose CdDiff, an image-based method that helps to visualize the results of mobile application compatibility testing for different devices, operative system versions, or resolutions.

2.5. *Actionable insights obtained from the systematic mapping of the literature*

The systematic mapping of the literature process, conducted to answer RQ1, has yielded insights leading to conclusions about the research fields covered by GUI change detection techniques and their mode of dissemination:

(1) Software testing change detection techniques for desktop, web, and mobile systems are frequently applied for purposes beyond specifically highlighting GUI delta changes.
(2) The terminology used to describe GUI change detection approaches is diverse across existing software testing studies, with varying adoption trends over the years.
(3) Research papers often lack explicit information about whether or how detected GUI changes are reported to users, such as script repair suggestions, bug change reports, or event flow graph models.
(4) Studies with ambiguous abstracts, inadequate summarization of content, or non-uniform terminology necessitate an in-depth snowball process for discovery.
(5) This field of research requires standardization by cataloging the objectives, terminology, practices, and challenges of delta GUI change detection.

The analysis of existing studies has allowed us to identify which technical implementations are valuable in preparing a change detection process. At the same time, it has helped us to uncover gaps in the current state-of-the-art, outlining the steps that require research and implementation efforts:

(1) State models or event flow graphs serve as conceptual artifacts to represent application state–action transitions. These models are valuable for implementing system-independent GUI change detection techniques. Additionally, they facilitate the visualization of action transitions, not solely focusing on states. This information can help developers and testers visualize the pathways leading to the changed GUI states.
(2) The report should be easy to understand for various users, as well as help visually by highlighting changes without using a large number of colors in an intrusive way.
(3) There are image comparison techniques that can potentially be used on diverse GUI systems. However, to the best of our knowledge, there is no proposal has been evaluated on GUI state–action transitions in three different systems, such as desktop, web, and mobile applications.

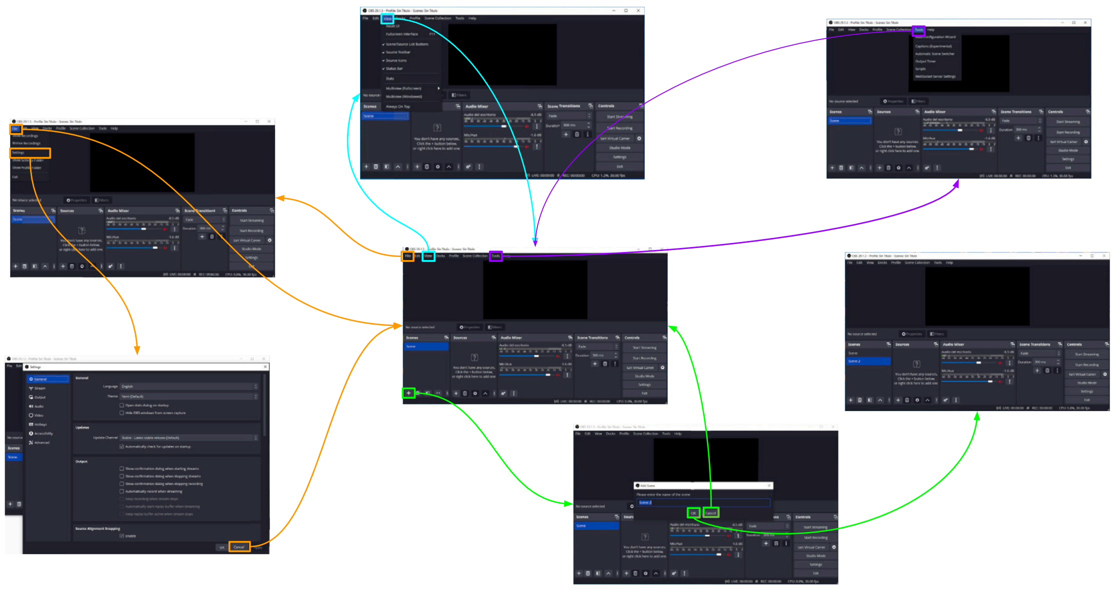


Fig. 4. Representation of a GUI state model of a desktop application that contains 7 states and 11 action transitions.

3. GUI state model inference

The Graphical User Interfaces (GUI) of desktop, web, and mobile applications consist of a set of widgets *w*. Each widget possesses a set of properties like its GUI position, a user-readable title, a role specifying whether it is a button, text field, or scroll bar, and its color for aesthetic visualization, among others. Furthermore, these widgets are commonly linked in a hierarchical structure known as *widget tree*. The *widget tree*, including all widget properties, determines the GUI state *s*.

While certain widgets are solely dedicated to presenting information to users, other widgets are interactive, enabling users to perform actions *a*. For example, actions on button widgets allow to trigger click events, text field widgets are typeable, and scroll bar widgets permit to slide dynamic panels, thus showing or hiding other widgets.

The state model inference process consists of automating the exploration of GUI systems by connecting to the System Under Test (SUT), detecting the state *s*, deriving all the possible actions to execute, and selecting and executing the action *a* in the origin state *s* to reach a target state *s'*. The entire set of discovered states *s*, derived actions *a*, and executed transition $s \rightarrow a \rightarrow s'$ constitutes the state model. Fig. 4 represents a GUI state model that contains 7 discovered states and 11 action transitions that connect them.

3.1. TESTAR tool for state model inference

TESTAR is an open-source scriptless tool that infers a state model while exploring and testing the SUT [59]. The tool employs various APIs and automation frameworks to connect and interact with different types of systems. Windows Automation API and Java access bridge are used for desktop applications [60], Selenium WebDriver for web pages [61], Appium for mobile applications [62], and external plugins such as iv4XR can be used for eXtended Reality (XR) systems [63,64].

We selected TESTAR to perform the GUI change detection research using inferred models because (1) the tool and the OrientDB graph database used to store the state model are open-source, (2) it supports the state model inference of diverse systems, (3) it has been evaluated with industrial and complex open-source applications [8,62,65], (4) is actively maintained, and, as we explain in the following sections, (5) it offers a set of Java protocols and settings files that allow defining an abstraction and inference strategy to deal with dynamism, non-determinism, and state space explosion challenges [60,61].

3.2. State model abstraction strategy

TESTAR uses the properties of the GUI widgets to identify *concrete* and *abstract* states in the state model inference process.

The *concrete* state identifier is calculated based on all the properties of all the widgets in the GUI. If any value of one of the widgets' properties changes, TESTAR identifies and stores a new concrete state in the model. This concrete technique allows precise detection and mapping of any changes in the GUI. However, for real and complex systems, the dynamic and highly modifiable properties of widgets make the concrete identifier a technique that generates extensive combinations of concrete states. For instance, Fig. 5 shows an example of widgets that dynamically alter their volume level, recording time, or CPU consumption properties to provide real-time information to users. Consequently, attempting a concrete identification of these states would result in an immense model with a prohibitive number of concrete states. This uncontrolled growth of models is known as a state space explosion [60,66].

To effectively manage state identification during the inference of complex systems, TESTAR calculates an *abstract* state identifier determined by a selected subset of the GUI widgets' properties. It is important to consider that TESTAR users need to define a suitable *abstraction strategy* to decide which widgets' properties should be included in the abstract state identifier. This abstraction strategy aims to try to encapsulate the fundamental variations in GUI states without succumbing to the intricacies introduced by their dynamic behaviors. Following the previous example in Fig. 5, an appropriate abstraction strategy could consist of considering the existence of the volume level, recording time, and CPU consumption widgets while ignoring their dynamic numerical values.

A similar technique is required for GUI actions. The *concrete* action identifier is calculated based on the *concrete* state on which the action was executed, the screen coordinates of the interacted widget, the role of the action (i.e., click or type), and the specific text if the action includes typing. In contrast, the *abstract* action identifier is calculated based on the *abstract* state and *abstract* identifier of the interacted widget, and the role of the action without considering the typed text.

3.3. State model inference strategy

When inferring a GUI state model, the idealistic goal could be to define an abstraction strategy capable of adequately identifying

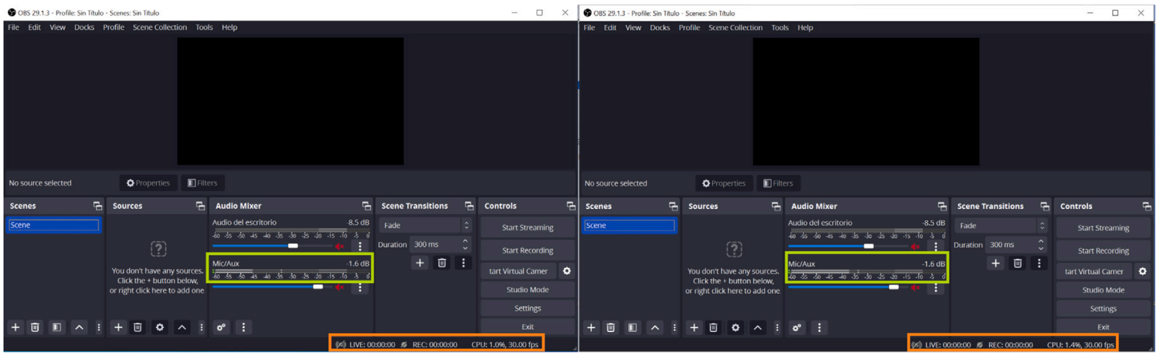

OBS 29.1.3 - Profile: Sin Título - Scenes: Sin Título
File Edit View Docks Profile Scene Collection Tools Help
No source selected
Properties
Filters
Scenes
Sources
Audio Mixer
Scene Transitions
Controls
Scene
Audio del escritorio
8.5 dB
Mic/Aux
-1.6 dB
Fade
Duration 300 ms
Start Streaming
Start Recording
tart Virtual Camer
Studio Mode
Settings
Exit
You don't have any sources. Click the + button below, or right click here to add one
LIVE: 00:00:00 REC: 00:00:00 CPU: 1.0%, 30.00 fps
LIVE: 00:00:00 REC: 00:00:00 CPU: 1.4%, 30.00 fps

Fig. 5. Dynamic widgets properties present in a GUI state.

all the states and actions within a system, representing the complete logic of transitions for a SUT. Nonetheless, achieving this goal proves unrealistic when applying GUI inference techniques to real systems. The extensive number of states and actions in large and complex systems and their intrinsic dynamic behavior pose a significant obstacle to inferring a complete model. It is essential to design an *inference strategy* that aligns with a manageable objective for state model inference.

In this research, we aim to infer various GUI state models for each delta version of the same application for subsequent GUI change detection analysis. Nevertheless, attempting to infer a complete model for a large and complex system is neither efficient nor realistic due to the essentials of the state explosion challenge [66]. Any technical mitigation or advanced solution that requires implementing an abstraction technique [67] or a test-driven strategy [68] may lack the inference of part of the SUT functionality. Therefore, we propose a solution that aims to balance the completeness of the inferred model while dealing with large models.

To accomplish our delta GUI change detection goal, we need a model that encapsulates all existing state–action transition functionalities. If some of these transitions are not inferred in the model, the detection of changes will provide false positives and negatives due to the lack of modeled GUI information. However, due to large and complex systems having the essential state explosion challenge, we assume the need to restrict the search space of the model to a manageable size. To achieve this objective, we infer a *partially complete model*. This partially complete model entails a limited set of actions with a restricted depth of state–action transitions.

For instance, Fig. 6 represents how the inference process of the model is restricted to a limited depth, represented as the encapsulated state–action transitions in the center of the image. The external states outside the partially complete model (i.e., the grayed state and action transitions outside the central encapsulated model) are also functional parts of the SUT that may require a high-depth exploration to be discovered and inferred in the model. The main objective of the inference strategy is to allow users to customize this limited set of actions with a restricted depth depending on the size and complexity of the model being created from the SUT.

3.4. State model abstraction and inference challenges

Defining the abstraction and inference strategies is an intricate task that requires a thorough understanding of two main challenges that affect the state model inference process: dynamism and non-determinism [61,66].

Dynamism poses challenges not only in handling the dynamic values of widget properties but also in addressing the presence of widgets that can be dynamically added or removed from the application's states. Interacting with these dynamic widgets can be important for verifying their correct functionality. However, attempting to capture the diversity of states and the combination of actions, as mentioned earlier, would lead to a state space explosion.

Fig. 7 illustrates a dynamism challenge on which widgets to add and remove scenes and recording sources alter the application states. First, this dynamic behavior provokes a state space explosion, generating new states and actions. This is because TESTAR continuously identifies new states due to the presence of newly introduced widgets. Second, if the inference process involves opening and closing the application to infer the state model, the resulting model would encompass a substantial number of initial states, representing the combinations of added/removed widgets.

Non-determinism presents another challenge that requires consideration during state model inference. A scenario is deemed deterministic when a specific action executed in a particular state consistently creates a transition to the same target state. However, an abstraction strategy that cannot track the dynamic widget behavior, lacks information in the GUI, or can be affected by external system factors may lead to an action transitioning to a different target state.

For instance, Fig. 8 illustrates a dynamic behavior that could induce non-determinism. Initially, a click action on the `Stream` widget in the `General settings` panel transits to the `Twitch service` panel. However, if an action changes the service to `Youtube`, a posterior click action on the `Stream` widget in the `General settings` panel will transit to the `Youtube service` panel instead of the `Twitch service` panel.

4. GUI change detection tool

The ChangeDetection tool[1] compares two inferred GUI state models from distinct software versions to detect and highlight GUI changes [12]. At first, a change detection algorithm simultaneously transits the corresponding states and actions of both state models to detect and mark the states that have been changed, added, or removed. Subsequently, a merged graph technique is employed to visualize the changed states, as well as the added and removed transitions. Fig. 9 shows two partial state models, SM_{new} (v30.0.2) and SM_{old} (v29.1.3), of two different versions of the OBS open-source desktop application.[2]

4.1. Change detection algorithm

The underlying idea of the ChangeDetection algorithm 1 is to recursively traverse transitions $s \rightarrow a \rightarrow s'$ in both SM_{new} and SM_{old} models while comparing the properties of the target states s'. As the identification of states and actions in the model relies on the chosen abstraction, the initial step in the algorithm validates that the models to be compared utilize the same abstraction properties (line 1). It is then that the algorithm finds the initial states of the models, establishes the initial corresponding state associations for the $newInitialState$ and

[1] https://github.com/TESTARtool/ChangeDetection.NET
[2] https://github.com/obsproject/obs-studio

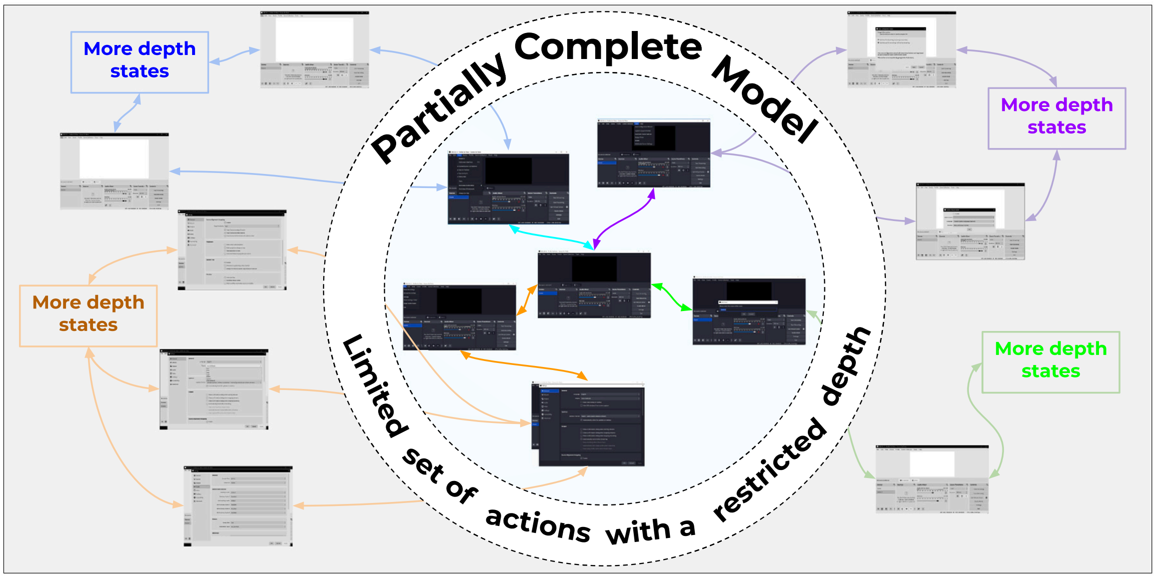


Fig. 6. Partially complete model inferred due to a limited set of actions with a restricted depth.

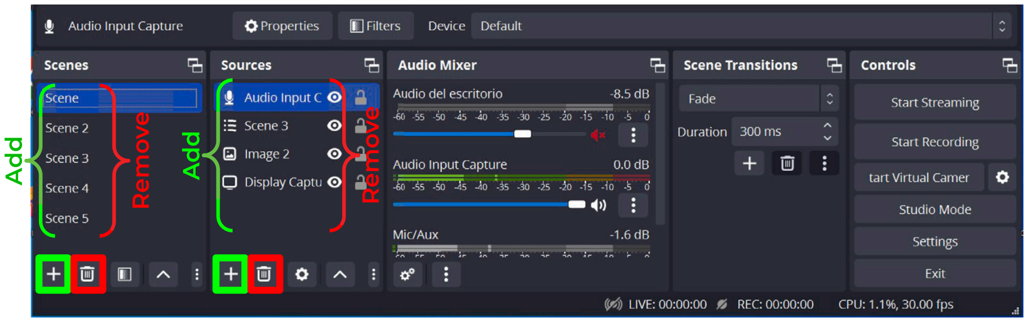
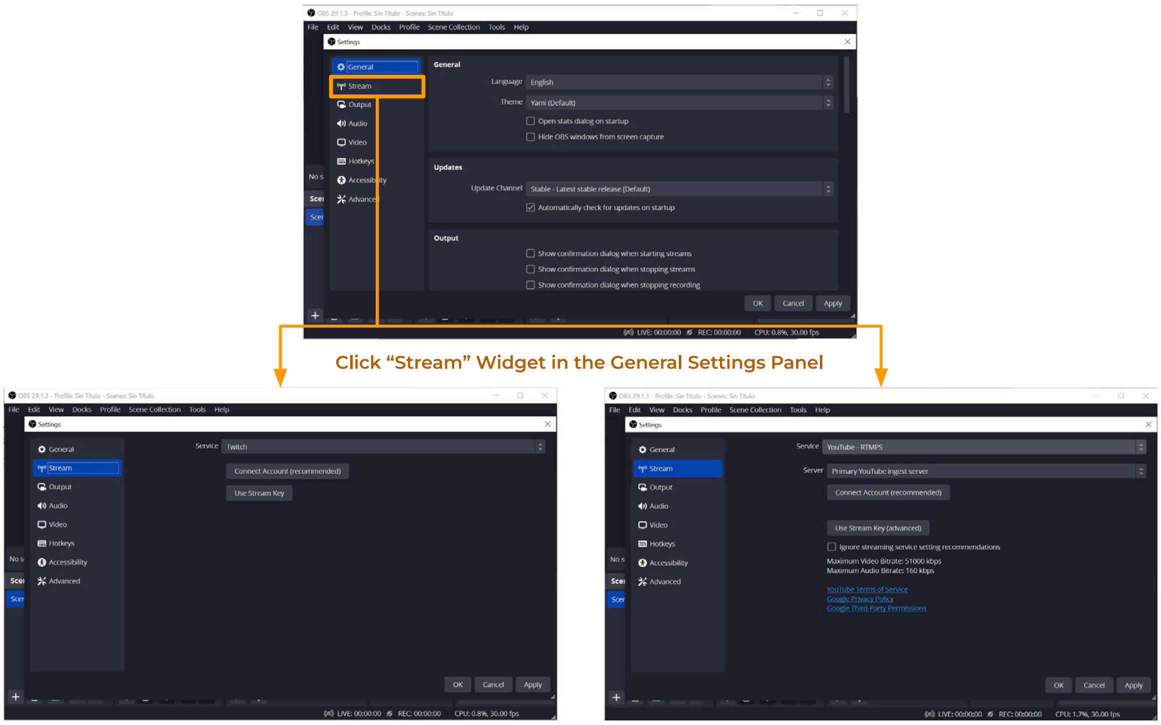


Fig. 7. Widgets that can be dynamically added or removed from the state.



General
Language English
Theme Yami (Default)
☐ Open stats dialog on startup
☐ Hide OBS windows from screen capture
Updates
Update Channel Stable - Latest stable release (Default)
☑ Automatically check for updates on startup
Output
☐ Show confirmation dialog when starting streams
☐ Show confirmation dialog when stopping streams
☐ Show confirmation dialog when stopping recording

Service Twitch
Connect Account (recommended)
Use Stream Key

Service YouTube - RTMPS
Server Primary YouTube ingest server
Connect Account (recommended)
Use Stream Key (advanced)
☐ Ignore streaming service setting recommendations
Maximum Video Bitrate: 51000 kbps
Maximum Audio Bitrate: 160 kbps
YouTube Terms of Service
Google Privacy Policy
Google Third-Party Permissions

Fig. 8. Potential non-deterministic behavior clicking the same widget, as different stream options may exist.

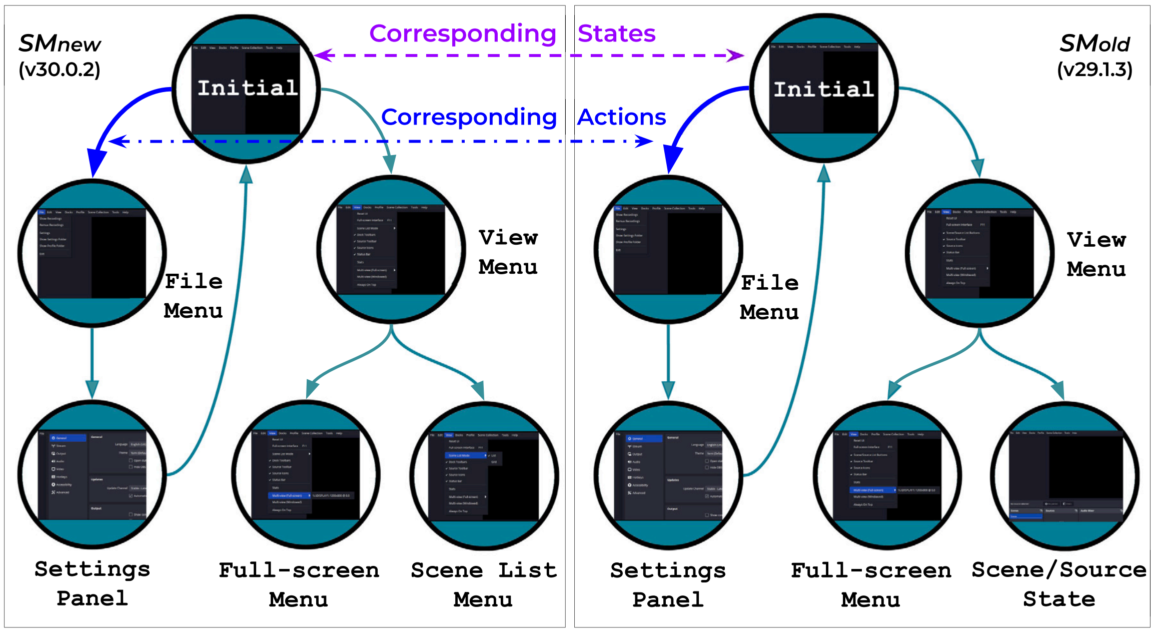


Fig. 9. OBS partial inferred state model from SM_{new} (v30.0.2) and SM_{old} (v29.1.3).

$oldInitialState$ (line 2), and starts to recursively transit the SM_{new} and SM_{old} models by invoking the `CompareStates` procedure (line 3).

The `CompareStates` procedure takes the corresponding $newState$ and $oldState$ as parameters for comparison (line 4). To prevent infinite recursion during model comparison, both corresponding states are marked as `Handled` (line 5). Then, the algorithm compares the state abstract identifiers of the corresponding states to detect whether the state has changed between versions (line 6). To continue traversing the model, the algorithm extracts a $newActionsList$ of unhandled actions for $newState$ (line 7) and a $oldActionsList$ of unhandled actions for $oldState$ (line 8). For each $newAction$ in the $newActionsList$ (line 9), the `CompareActions` procedure is invoked (line 10).

The `CompareActions` procedure takes a $newAction$ from $newState$ and the $oldActionsList$ of the corresponding $oldState$ as parameters (line 13). Similar to the `CompareStates` procedure, the $newAction$ is marked as `Handled` to prevent infinite model recursion (line 14). First, the algorithm checks if the $newAction$ has a corresponding $correspondingOldAction$ in the $oldActionsList$ (line 15). If $correspondingOldAction$ is null (line 16), indicating the absence of a corresponding action in SM_{old}, the algorithm marks $newAction$ as a new model transition and finishes the recursive comparison in this part of the model (line 17). Conversely, if a $correspondingOldAction$ is found, the algorithm marks $newAction$ as a matched transition (line 19) and marks $correspondingOldAction$ as `Handled` (line 20). To continue traversing the model, the algorithm retrieves the $newTargetState$ of the $newAction$ (line 21) and the $oldTargetState$ of the $correspondingOldAction$ (line 22). Finally, if the subsequent $newTargetState$ and $oldTargetState$ have not been handled yet (line 23), the `CompareStates` procedure is invoked to continue the recursive model comparison (line 24). If the subsequent states have already been compared and handled, the recursive comparison concludes in this part of the model.

The ChangeDetection algorithm 1 traversed the models, comparing the identifiers of corresponding states and actions and marking them as matched or new. However, certain states and actions from both SM_{new} and SM_{old} models may remain uncompared due to the difference in the abstract identifiers within the model paths. In the subsequent merge graph technique, the remaining states and actions from SM_{new} that were not compared are considered as *new*, while those from SM_{old} that were not compared are designated as *removed*.

4.2. Merge graph technique

The merge graph technique allows two graphs to be merged into one graph for further visualization and analysis [69]. After the ChangeDetection algorithm has transited the corresponding states and actions of both SM_{new} and SM_{old} models, this technique creates a merged graph model SM_{merged} and executes two merging steps that can be visualized in Fig. 10:

(1) Add all states and actions from SM_{new}.
The model SM_{new} contains three possible categories of states after the execution of the ChangeDetection algorithm.
First, the corresponding states remain identical in both delta versions (e.g., `Initial`, `File Menu`, and `Settings Panel`). These identical states are visualized with opaque circles.
Second, the corresponding states contain changes between delta versions (e.g., `View Menu`). These changed states are visualized with a dashed linear border.
Third, there are state-transitions that existed in SM_{new} and in which the ChangeDetection algorithm has not found a corresponding state in SM_{old} (e.g., `Full-screen Menu - new` and `Scene List Menu`). These are newly added state-transitions visualized with a green star.

(2) Add non-matching states from SM_{old} and wire actions.
The state-transitions that were not handled in SM_{old} during the ChangeDetection algorithm are state-transitions that do not exist in SM_{new}. Hence, these are removed state-transitions between delta versions (e.g., `Full-screen Menu - old` and `Scene/Source State`). These are old removed state-transitions visualized with a red triangle.

The SM_{merged} serves as an interactive model, enabling users to choose specific states and actions for visualization and analysis of change detection results. Green stars and red triangles show screenshots of the newly added or old removed states, respectively. Circle states, which indicate identical abstract states, display two screenshots—one for the new state and one for the old state. Finally, changed states represented by dashed linear borders, in addition to the two screenshots

Algorithm 1 ChangeDetectionAlgorithm

```
Require: $SM_{new}$                                   ▷ The inferred state model from the new application version
Require: $SM_{old}$                                   ▷ The inferred state model from the old application version
 1: CheckEqualAbstractAttributes($SM_{new}$, $SM_{old}$)   ▷ Both state models must be using the same abstract properties
 2: ($newInitialState$, $oldInitialState$) = FindInitialStates($SM_{new}$ , $SM_{old}$)   ▷ Initialize corresponding states
 3: CompareStates($newInitialState$, $oldInitialState$)   ▷ Invoke the state comparison procedure
 4: procedure CompareStates($newState$, $oldState$)
 5:     MarkHandledStates($newState$, $oldState$)   ▷ Mark states as handled
 6:     CompareStateIdentifiers($newState$, $oldState$)   ▷ Compare state identifiers to detect changes
 7:     $newActionsList$ = FindUnhandledActions($newState$)   ▷ Retrieve the actions for the new state
 8:     $oldActionsList$ = FindUnhandledActions($oldState$)   ▷ Retrieve the actions for the old state
 9:     for $newAction \in newActionsList$ do   ▷ Iterate through each new action to
10:         CompareActions($newAction$, $oldActionsList$)   ▷ Invoke the actions comparison procedure
11:     end for
12: end procedure
13: procedure CompareActions($newAction$, $oldActionsList$)
14:     MarkHandledAction($newAction$)   ▷ Mark the new action as handled
15:     $correspodingOldAction$ = FindCorrespondingAction($newAction$ , $oldActionsList$)   ▷ Find the corresponding action
16:     if $correspodingOldAction$ is NULL then   ▷ If there is not a corresponding action in the old model
17:         SetTransitionAsNew($newAction$)   ▷ The new action is a new transition
18:     else   ▷ If a corresponding action exists in the old model
19:         SetTransitionAsMatch($newAction$)   ▷ The new action is an existing matched transition
20:         MarkHandledAction($correspodingOldAction$)   ▷ Mark the corresponding old action as handled
21:         $newTargetState$ = GetState($newAction$)   ▷ Get the target state of the transited new action
22:         $oldTargetState$ = GetState($correspodingOldAction$)   ▷ Get the target state of the transited old action
23:         if NotHandledStates($newTargetState$, $oldTargetState$) then   ▷ If those states were not handled
24:             CompareStates($newTargetState$, $oldTargetState$)   ▷ Invoke the next states comparison procedure
25:         end if
26:     end if
27: end procedure
```

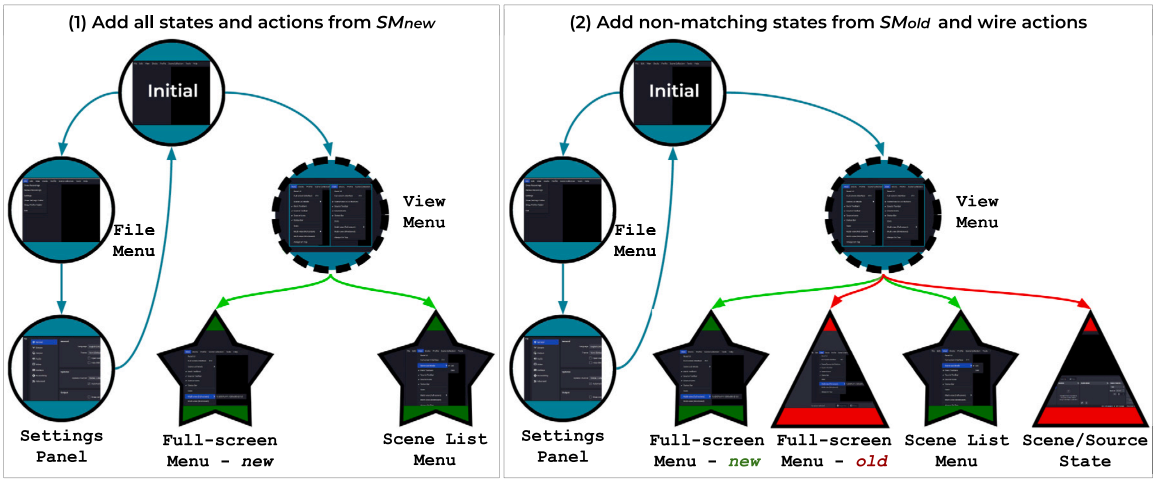


Fig. 10. Merge graph technique for visual graph comparison with OBS partial inferred state models.

of the new and old states, generate a third screenshot created using a pixelmatch library[3] to highlight the GUI changes (see Fig. 11).

4.3. Configurable action abstraction identifiers

In the merged model from Fig. 12-A, the transition to the `Full-screen Menu` state is detected at the same time as added and removed state-transitions. This is provoked because the abstraction strategy for identifying actions relies on the abstract identifier of the origin `View Menu` state, which has undergone changes.

In our prior research [12], we observed that this behavior may not only result in a visually confusing representation for users but can also impede the effectiveness of the ChangeDetection approach. If the corresponding state associations disappear due to a state-transition originating from a changed state, all subsequent transitions from the `Full-screen Menu` state will be incorrectly regarded as entirely newly added or removed.

State models can employ descriptions to identify action transitions $s \rightarrow a \rightarrow s'$ (e.g., *Left click at 'Click Full-screen'*). This action description can remain consistent even if the abstract identifiers of the origin states have changed between versions. For this reason, we opted to extend the ChangeDetection tool with a configurable option that enables users to decide if the tool needs to use the action abstract identifier or the action description.

[3] https://github.com/mapbox/pixelmatch

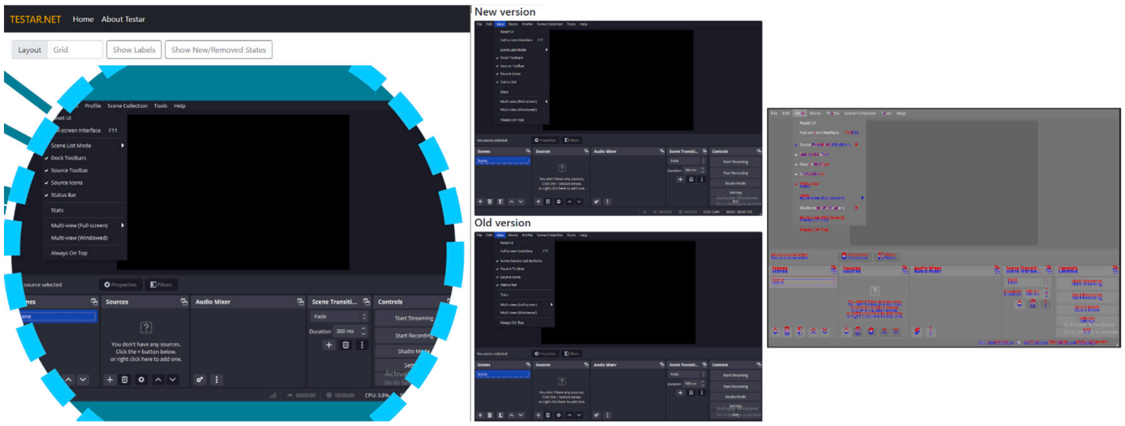


Fig. 11. Visual representation of changed states.

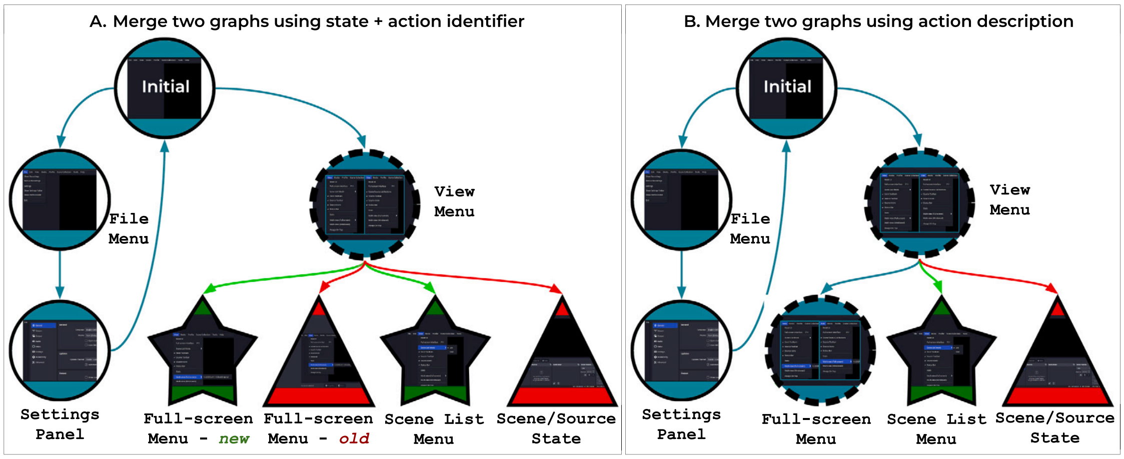


Fig. 12. Merge graph technique comparison using state and action abstract identifier vs. action description.

Fig. 12-B illustrates this configurable alternative for the same merged model. In this case, the tool detects that the action description of *Left click at 'Click Full-screen'* remains consistent across both OBS versions. As a result, the `Full-screen Menu` is marked as a changed state. This involves considering the `Full-screen Menu` state as corresponding states in the traverse algorithm, potentially enabling subsequent transitions to be compared.

5. Empirical study

The objective of this study is to evaluate the importance of using a GUI change detection approach to facilitate the identification and visualization of delta GUI changes. This will help to validate that the GUI changes have occurred as intended or to reveal unforeseen GUI modifications. To achieve this, we intend to verify whether the automated utilization of state model inference through a scriptless tool, followed by the application of the ChangeDetection tool, effectively highlights changed, added, and removed states in delta changes of software projects.

In order to guide the study, we have formulated the following research question:

- `RQ2`: Does the ChangeDetection tool detect delta GUI changes when using inferred state models?

To answer `RQ2`, we designed a controlled experiment following the guidelines proposed by Wohlin et al. [70] and aligned with a methodological framework specifically built to evaluate software testing techniques [71]. For the purpose of validating the tool with diverse application systems and disseminating the results, we have opted to select one desktop, one web, and one mobile open-source system.

This experiment entails the inference of multiple GUI state models from iterative versions of the same system, followed by the application of the ChangeDetection tool to each pair of model versions within the same system. For each ChangeDetection comparison between version pairs, we quantify and verify the number of detected GUI changes.

To analyze the change detection results and be able to answer the research questions, we defined the following null hypothesis:

- H_0: The ChangeDetection tool does not detect delta GUI changes using inferred state models.

5.1. SUT objects

The SUTs selected for this experiment, suitable for the change detection approach, must be published under an open-source license to allow software analysis and facilitate the replication of the experiments. Moreover, the TESTAR tool should be able to connect and detect the GUI widgets in order to infer a state model. Based on these open-source requirements, we selected the following SUTs:

- OBS Studio [72] is an open-source desktop application designed for capturing, recording, and streaming video content. At the end of 2023, it is the 6th most-starred C code software project on GitHub. It is widely known by various categories of users (e.g., teachers, event organizers, streamers, etc.) and is actively under development and maintenance.

Table 2
Details of the selected SUT Objects.

Metric	OBS-Studio	Calibre-Web	MyExpenses
SUT type	Desktop	Web	Android
GitHub Stars	52,700	10,600	630
GitHub Issues	3,300	2,300	1,100
Contributors	575	198	56
LOC	647,352	270,920	249,021
SUT Versions	v27.2.4 (Mar 30, 2022)	v0.6.18 - Suleika (Apr 3, 2022)	v3.6.6 - r657 (Oct 30, 2023)
	v28.1.2 (Nov 5, 2022)	v0.6.19 - Daria (Jul 31, 2022)	v3.6.7 - r665 (Nov 13, 2023)
	v29.1.3 (Jun 19, 2023)	v0.6.20 - Ella (Mar 27, 2023)	v3.6.8 - r671 (Nov 21, 2023)
	v30.0.2 (Dec 10, 2023)	v0.6.21 - Romesa (Oct 21, 2023)	v3.6.9 - r678 (Dec 4, 2023)

- Calibre-Web [73] is an open-source web application that offers a clean and intuitive interface for browsing, reading, and downloading eBooks. It is also a widely known application and is actively under development and maintenance.
- MyExpenses [74] is an open-source mobile Android application designed to keep track of user expenses and incomes and to export them in different file formats. Even though it is the least GitHub-starred of the three systems, the application has been in active development and maintenance for more than 10 years.

The details obtained at the end of 2023 regarding the selected SUTs are presented in Table 2. The lines of code (LLOC), together with the GitHub stars, issues, and contributors, indicate that these are not toy software projects, i.e., they are representative SUTs of real and widely adopted applications. For each SUT, we selected 4 recent versions to evaluate the change detection approach. We consider the first selected version the control baseline, and the following versions will be used to evaluate the detect GUI delta changes [70].

The TESTAR tool can be employed for testing purposes through the implementation of diverse oracles that verify the presence of failures in the system's GUI. However, given that this study does not focus on the validation of the scriptless technique for fault detection, we have opted to apply the blocking principle [70] to disable TESTAR oracles.

5.2. Independent variables: Inference strategy

The objective of this study is to validate if the ChangeDetection tool is capable of detecting and reporting changes between SUT versions. To accomplish this objective, it is necessary to infer a partially but complete state model that represents all the existing GUI transitions of a section of the SUT. If the inference process is not restricted and controlled, the obtained state model may result too big and complex, potentially inducing a wide variety of dynamism, non-determinism, and state explosion challenges. For this reason, we defined a set of independent variables regarding the inference depth limit and the actions to derive and execute in the inference process.

First, we designed a similar **sequences inference strategy** (see Table 3) based on re-launching the SUT Objects multiple times and limiting the maximum depth of the models. Running a large number of sequences of 3 action lengths reduces the number of discovered states but facilitates the inference of the partially but complete state model. Additionally, it mitigates abstraction challenges that may arise and makes it easier for users to define an appropriate abstraction strategy.

Another important inference factor to consider is the time duration given to the actions to execute the GUI-events interactions and the time to wait between the execution of actions. The action duration of 0 s, indicates the TESTAR tool that it is mandatory to teleport the mouse to the interactable widget coordinates and perform the GUI-event interaction. This is important in the inference process of desktop and web GUI systems since mouse movements instead of mouse teleport can trigger mouse-over GUI events and provoke unintentional GUI alterations. Finally, because applications require time to load widgets completely in the GUI, we decided to give all of them 2 s after each action execution.

Table 3
Independent variables for sequences inference strategy.

	Depth limit	Action duration	Wait after action
OBS-Studio	3 actions	0 s	2 s
Calibre-Web	3 actions	0 s	2 s
MyExpenses	3 actions	0 s	2 s

Second, we configured an **action derivation strategy** (see Table 4) aiming to control the execution of actions and potentially reduce the abstraction inference challenges:

- *Force* actions are specific step-by-step actions necessary to start the SUT in the initial desired state.
- *Click* actions indicate with which type of widgets we focus on click-interacting in the model inference process.
- *Filter* actions are used to ignore widgets that open external system processes, ignore dynamic or high-combinatorial widgets that increase the state model size, and ignore detected widgets that create non-deterministic transitions.
- *Kill* actions are intended to detect and close web browser and file explorer processes that appear during the inference process. These are implemented explicitly for the OBS desktop SUT because it is a widely used functionality in various widgets, and defining rules to filter all these widgets would require a lot of effort and maintenance over versions.

The *Force* actions executed to start the model in the initial state (e.g., login, close update initial panel) are not inferred in the model and are not considered part of the maximum depth limit. However, this is an inference disadvantage in this study, since states forced before reaching the desired initial state are not tracked in the model for subsequent change detection.

Typing actions are essential to interact with the SUT and manage OBS scenes, Calibre books, or money Expenses. Nevertheless, since the inference strategy focused on clicking to discover existing menus, lists, and configuration panels already provide sufficient states and actions in the models to perform the change detection evaluation, we decided not to derive typing actions.

Sliding actions are challenging for the state model inference process since they change the visible and interactable widgets in the GUI. This affects the abstract strategy and increases the complexity of the exploration space. For this reason, we decided not to derive sliding actions.

5.3. Independent variables: Abstraction strategy

The abstraction strategy consists of implementing a main abstraction mechanism and a set of sub-strategies. The main abstraction mechanism uses widget properties with the objective of distinguishing the abstract states and actions in the state model that represents the SUT behavior. Table 5 shows the widget properties used for OBS-Studio, Calibre-Web, and MyExpenses.

Customizing the depth limitations and action derivation in the inference strategy reduces the inference complexity and the need to

Table 4
Independent variables for action derivation strategy.

OBS-Studio	(Force) Close the update panel that appears in the GUI when a new version is available to be installed (Click) Focus on MenuItem and ListItem widgets to infer a partially complete model of the configuration menus. This also prevents interacting with dynamic widget-icons (see Fig. 7) (Click) Close or Cancel buttons if there are no MenuItem and ListItem widgets in the state (Filter) Widgets that open the update version panel and widgets that, in a non-deterministic amount of seconds, verify the integrity of the files (Kill) Web Browser and File Explorer processes that are invoked when interacting with video and audio management widgets
Calibre-Web	(Force) Login with valid credentials to start in the initial GUI state as an admin user (Click) All the various types of clickable web widgets in this SUT (hyperlinks, buttons, checkboxes) (Filter) One specific List-grid widget that provoked non-determinism when transiting again to the state (similar to Fig. 8) (Filter) Widgets that logout, and widgets that save or download files
MyExpenses	(Force) Skip the initial configuration and focus on opening the wallet, plus another action that opens the manage accounts menu (Click) All widgets with enabled and clickable Android attributes (Filter) Menu widgets in the initial state that can be shown/hidden. This provokes non-determinism and, trying to address it, a significant increase in the size of the model (Filter) Widgets that open states with completely dynamic date and hours information (Filter) Widgets with enabled and disabled switch functionality that increase the size of the model due to the combinatorial possibilities (Filter) Tell-a-friend widget that opens an Android system menu trying to send an email

Table 5
Independent variables for the main abstraction mechanism.

OBS-Studio	
State	Path, ControlType
Action	OriginState + OriginWidget + ActionRole
Calibre-Web	
State	WebId, WebTextContent
Action	OriginState + OriginWidget + ActionRole
MyExpenses	
State	AndroidXPath, AndroidText, AndroidClassName
Action	OriginState + OriginWidget + ActionRole

deal with abstraction challenges. Even so, in complex systems, it can be necessary to design a set of abstraction sub-strategies that complement the main abstraction mechanism to enhance the abstract states and actions identification or to deal with dynamism and non-determinism.

Table 6 shows the following implemented types of abstraction sub-strategies:

- *Added* properties of specific widgets to the main abstraction mechanism. This is necessary if the default property values are not enough to differentiate widgets from each other.
- *Ignore* widgets that can be dynamically added or removed from the state during the inference or dynamic properties of specific widgets.
- *Event* SUT behaviors that may provoke non-determinism in the model during the inference process need to be addressed with special triggered actions.

The OBS-Studio and MyExpenses SUT objects are distinguished in their inverse abstraction strategies. In the case of OBS-Studio, we decided to omit the Title property in the main abstraction mechanism since various widgets have dynamic Title values. Then, we incorporated the Title property of MenuItem, ListItem, CheckBox, and ComboBox widgets as an abstraction sub-strategy. Conversely, in the case of MyExpenses, we decided to include the AndroidText property in the main abstraction mechanism and custom abstraction sub-strategies to ignore the dynamic widgets. Both options can be valid to implement, and they also depend on the depth limit and derived action from the inference strategy, or the complexity of the functionalities of the SUT.

Caliber-Web did not require various abstraction sub-strategies because the combination of the inference strategy and the main abstraction mechanism was adequate.

5.4. Effort time for independent variables

Deploying and analyzing the versions of the SUT objects to custom these independent variables requires effort to identify, configure, and validate their adequacy. The proficiency obtained from previous research in learning to design inference and abstraction strategies for desktop and web systems allowed us to reduce the effort time in this study. Principally, it helped the decision to prepare an inference strategy that restricts the depth to infer the partially complete models [12]. Table 7 contains an approximation of the times required to manually configure and refine the inference and abstraction strategies, as well as the automated pre-execution time it took to infer state models from different versions to validate the partially complete models where inferred adequately.

OBS-Studio required approximately 10 h of manual analysis combined with 8 h of automated pre-executions to obtain the appropriate configuration. Most of the effort was invested in determining the best action derivation and abstraction sub-strategy to deal with the opening of the web browser and file explorer and deciding to completely ignore the dynamic stats widgets.

Calibre-Web required approximately 6 h of manual analysis combined with 4 h of automated pre-executions to obtain the appropriate configuration. In the initial analysis, we recognized the robust implementation of the WebId property along the different versions of the SUT. The widget that provoked non-deterministic was the unique encounter challenge we decided to filter within the action derivation strategy.

MyExpenses required approximately 15 h of manual analysis and 25 h of automated model inference pre-executions to decide about an appropriate configuration. In this SUT, it was necessary to perform more preliminary experiments to find an appropriate balance with the inference and abstraction strategies to control the size of a deterministic model.

5.5. Dependent variables

To answer RQ2:, we performed a qualitative evaluation [71]. This evaluation is manually performed by an expert in GUI testing with more than five years of experience, which aims to validate whether the detected states marked as changed indeed contain GUI changes. Subsequently, we need to check if certain SUT functionalities or displayed GUI changes do not correspond with the delta SUT version changes. Finally, it is important to examine the results obtained from these complex SUTs to refine the process for further improvements. To accomplish this, we analyze the detected delta GUI changes with the related open-source software versions. In this way, we will evaluate the following:

(1) True positives change results: The state model inference and the change detection process correctly detect GUI changes that align with software delta version changes.

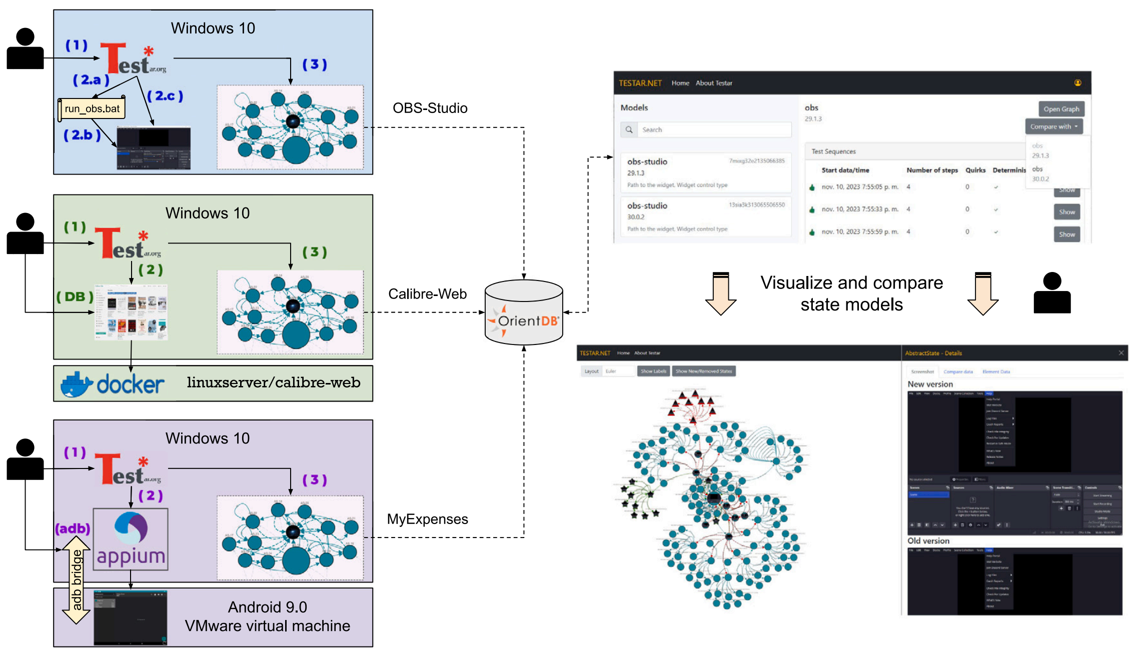


Fig. 13. Architecture of GUI delta change detection experiments.

Table 6
Independent variables for abstraction sub-strategies.

OBS-Studio	(Added) The Title property for MenuItem, ListItem, CheckBox, and ComboBox widgets since these widgets are not dynamic with a depth limit of 3 actions. Moreover, we aim to detect if those Title properties have changed between SUT versions (Added) The identifier of the origin state on which the (Kill) Web Browser and File Explorer actions are performed (Ignore) All dynamic widgets from the below usage status bar and from the panels that show the usage CPU and MEM stats (Event) Force a teleport of the mouse to the top-left coordinates of the screen (1) at the beginning of each sequence or (2) if, after executing an action, the mouse coordinates over a widget triggers a pop-up tooltip message that remains around 10 s
Calibre-Web	(Event) Force a teleport of the mouse to the top-left coordinates of the screen at the beginning of each sequence
MyExpenses	(Added) The Checked property of CheckedTextView widgets to prevent non-determinism when validating a form and transit to a destination state (Ignore) The dynamic AndroidText property of widgets that contain a date (`HH:MM AM/PM`) or hours (`DD/MM/YY`) patterns (Ignore) The dynamic AndroidText property of widgets from a Calculator (Ignore) The dynamic AndroidText of dropdown spinner widgets when they are closed, as they can create a high combination of possibilities. These dropdown spinners are not (Filter) such as the switch widgets, because we want to detect changes in the values when they are open. (Event) Wait 10 s if, after executing an action, the state contains a temporal snackbar message widget that remains around 5 s. In this system, it is not possible to just move the mouse away.

Table 7
Effort time to design the independent variables and run pre-executions.

	OBS-Studio	Calibre-Web	MyExpenses
(Manual) Strategies configuration	10 h	6 h	15 h
(Automated) Pre-executions	8 h	4 h	25 h

(2) Challenging results: The state model inference and the change detection process indicate the detection of GUI changes that do not align with the expected results.
(3) Further improvements: The state model inference and the change detection process require improvements to detect and highlight delta GUI changes with complex SUTs.

5.6. Design of the experiments

We use the TESTAR tool with the independent variables configurations to infer a state model for each delta version of a SUT. This inference outputs 4 state models for each SUT. Then, we apply the ChangeDetection tool with the pairs of delta state models of the same SUT. This change detection comparison reports 3 merged state models with detected changes for each SUT.

Fig. 13 shows the overview of the architecture. The inference process is similar for all SUTs, but because each system is inherently different, we explain their distinction execution below:

- **OBS-Studio:** This SUT is a portable Desktop application that does not require any installation process in a Windows environment. When TESTAR is launched `(1)`, an intermediate batch script `(2.a)` changes to the specific OBS directory and launches the application executable `(2.b)`. Then, TESTAR is able to connect to the OBS process `(2.c)` and automatically explore the SUT to infer the state model `(3)`. The resolution in the Windows 10 machine that renders the GUI is 1200 × 800.
- **Calibre-Web:** This SUT is a Web application that can be deployed using a Docker version image. Before launching TESTAR, it is necessary to configure a Calibre database to store the user books. In our experimentation, we deployed the Docker image in an Ubuntu environment and manually selected the Calibre database offered in the GitHub repository `(DB)`. Then, when indicating the web URL to TESTAR `(1)`, the tool is able to connect to the SUT `(2)`, and automatically explore the SUT to infer the state model `(3)`. The resolution in the Windows 10 machine that renders the GUI is 1200 × 800.
- **MyExpenses:** This SUT requires an Android system to be deployed. We created an Android 9.0 virtual device in a VMware

Table 8
state model inference results.

State model	Abstract states	Abstract actions	Inference time
OBS-Studio v27.2.4	132	319	2 h 5 m
OBS-Studio v28.1.2	132	265	1 h 40 m
OBS-Studio v29.1.3	132	198	1 h 10 m
OBS-Studio v30.0.2	134	206	1 h 10 m
Calibre-Web v0.6.18	25	436	1 h 25 min
Calibre-Web v0.6.19	35	499	1 h 50 min
Calibre-Web v0.6.20	37	499	1 h 50 min
Calibre-Web v0.6.21	37	499	1 h 50 min
MyExpenses v3.6.6	123	243	1 h 50 min
MyExpenses v3.6.7	125	246	1 h 55 min
MyExpenses v3.6.8	124	248	2 h
MyExpenses v3.6.9	125	248	2 h

environment and connected it with the Windows 10 host that contains TESTAR by creating an `adb bridge`.[4] When TESTAR is launched (1), the Appium[5] server detects the connected device and acts as middleware to obtain the state information (2). This permits TESTAR to automatically explore the SUT to infer the state model (3). The resolution of the Android 9.0 virtual device that renders the GUI is 1024 × 720.

At the end of the GUI state model inference processes, users can deploy the ChangeDetection tool using one Docker server image and one Docker client image. By specifying the IP and port address of OrientDB, users can inspect the state models individually and execute the comparison feature to perform the ChangeDetection algorithm. The results of the algorithm are automatically used by the merge graph technique, providing users with visualization, manual interaction, and analysis of GUI change detection results.

6. Results

This section presents the state model inference and change detection approach results. First, we summarize the inference process and size of the obtained GUI state models using the aforementioned independent variables configurations. Second, we present and discuss, for each SUT object, the results of comparing each delta version with the corresponding baseline to examine the qualitative evaluation for GUI change detection. The replication package with the state models and a complete document with visual results can be found here[6].

6.1. State model inference

Table 8 presents an overview of the inferred GUI state models, emphasizing the size in terms of discovered abstract states and executed abstract actions. Additionally, we provide information about the approximate time it took to infer the partially complete models for each SUT. The size of the models varies depending on the depth limit of the inference strategy and the properties information customized in the abstraction strategy. The inference time may vary depending on the action duration, wait after action, and force login or starting actions adopted in the inference strategy (i.e., TESTAR waits 10 s when starting OBS, and TESTAR takes around 10 s to perform the MyExpenses login). Although the model size and inference time are not directly related to the delta GUI change detection evaluation, these are essential aspects when considering techniques for evaluating rapid software delta increments.

For each SUT object, the same independent variables configuration has been used to infer the delta GUI state models. The inference and abstraction strategies for the GUI state models inference have remained resilient, requiring no adjustments to be adapted to the appearance of new dynamic widgets or those inducing non-deterministic behaviors. Nonetheless, as we mention in the following change detection results, the appearance of new widgets disrupts with *noise* the change detection technique. This *noise* is produced when widgets are either newly introduced, removed, or changed within the main static menus and panels of the SUT.

6.2. OBS-studio results

Fig. 14 shows an overview of how the OBS merged results are reported to users through an interactive web interface. This merged model includes distinct visual elements. Opaque green circle states signify OBS states that persist unchanged across delta versions. Dashed states that contain small images are changed states from OBS that have undergone GUI modifications. Red triangle transitions represent states and actions that have been removed in the delta software increment, while green star transitions highlight newly added states and actions. Clicking on any of these elements displays an informative panel on the right side of the web interface, presenting detailed images and textual information that users can use for the analysis and understanding of GUI changes.

Table 9 summarizes the change detection results. The change detection algorithm for OBS-Studio has been executed using the `action description`. The states that have not been reached due to state model inference restrictions (action derivation and filters strategy, or depth limit) are not included in the table. Furthermore, the states that contain changes ignored by the abstraction mechanism, such as the version and contributors texts changed in the `About` panel, are also not included in the table.

Based on the evaluation of the results obtained with this complex OBS-Studio desktop application, we can consider the challenging functionalities that negatively affect the state model inference and change detection process, as well as the further improvement ideas for the analysis of the detected changes.

6.2.1. OBS-studio challenging results

In the delta version from v27.2.4 to v28.1.2, the main `Audio Mixer` and `Scene Transitions` dock panels affect the change detection process of all the SUT states. Similarly, this occurs in the delta version from v28.1.2 to v29.1.3 with the new `Open Scene Filters` icon-button added in the `Scenes` dock panel.

These new widgets introduced in the main static panels do not interfere with the state model inference process itself but introduce change detection *noise* in all the states of the merged model (see Fig. 15). This change detection *noise* implies that all states will be marked as changed. Then, the user needs to analyze all states to determine if there are changes or not.

6.2.2. OBS-studio further improvements ideas based on results

Given alterations in GUI style, menus, and panel dimensions, using the pixelmatch technique proves insufficient in effectively assisting users in visualizing specific GUI changes. For instance, as illustrated in the previous Fig. 11, the pixelmatch technique highlights changes when the GUI widget coordinates have a slight deviation due to style changes, even if the widgets are the same with identical properties. A visual improvement can consist of comparing the abstraction identifier of the widgets of the two changing states, specifically detecting discordant widgets and using their position on the screen to highlight them with a rectangle.

In the delta version from v29.1.3 to v30.0.2, the **Settings-Advanced panel** contains a new `IP Family` change in the widget-tree that is not directly visible to the user when inspecting the merged graph. It has been necessary a complementary GUI and code user analysis to validate the existence of this delta change. A descriptive message indicating the changed widget together with the visual screenshot, could help users to recognize the changes easily.

[4] https://developer.android.com/studio/command-line/adb
[5] https://appium.io/
[6] https://doi.org/10.5281/zenodo.13712379

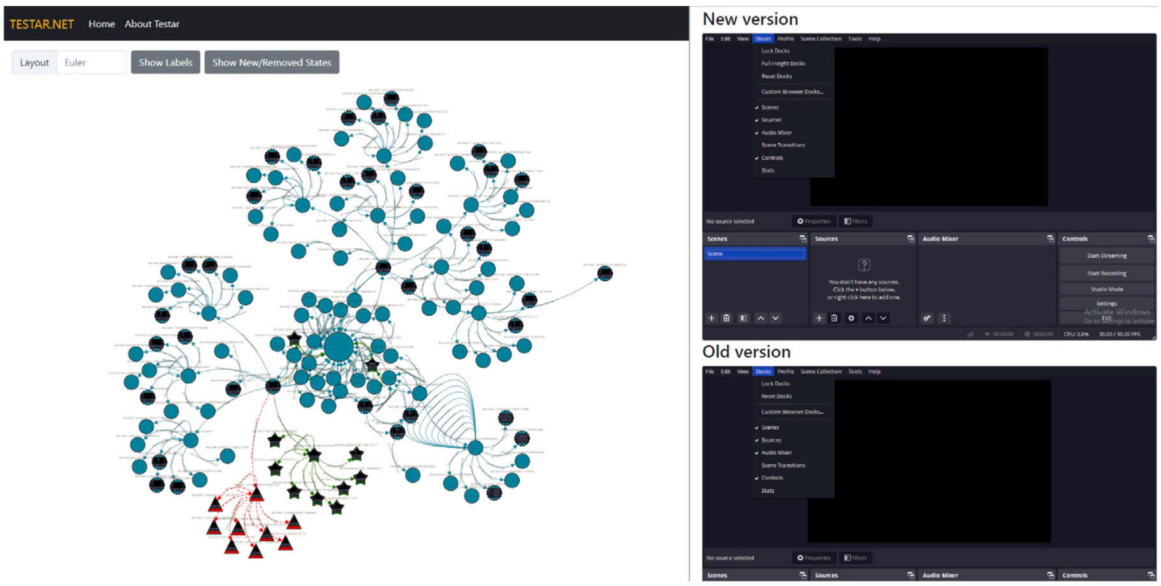


Fig. 14. Interactive web interface of the GUI change tool that creates a merged partially complete model from the OBS SM_{new} (v30.0.2) and SM_{old} (v29.1.3) versions with a depth of 3 actions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

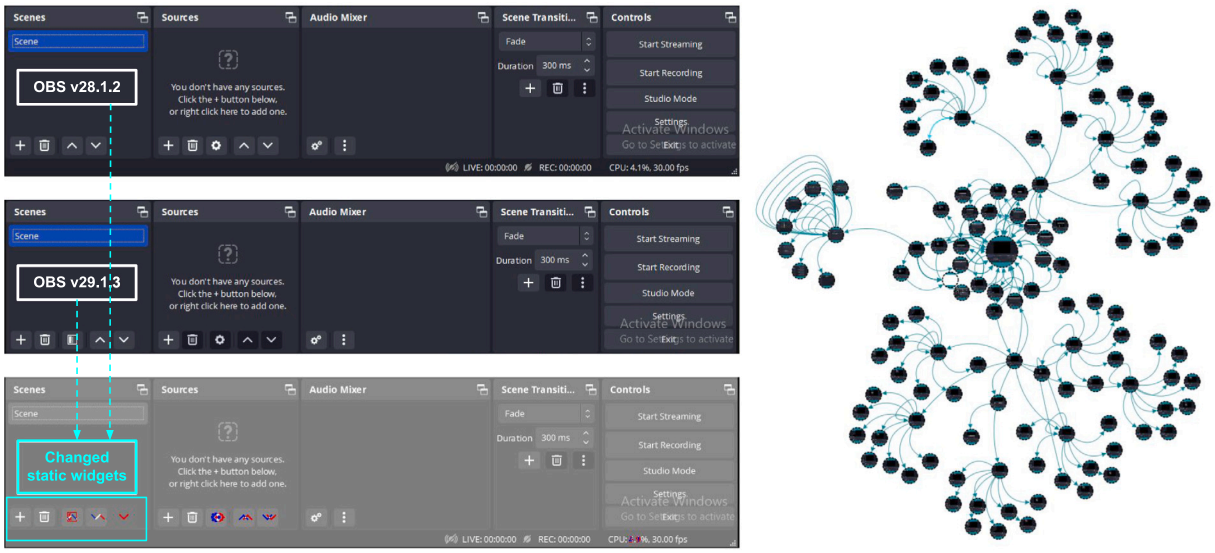


Fig. 15. Static widget changes challenge: The new `Open Scene Filters` icon-button added in the `Scenes` dock panel from v28.1.2 to v29.1.3 introduces *noise* in all states since they are all marked as changed.

6.3. Calibre-web results

Table 10 summarizes the change detection results. The change detection algorithm for Calibre-Web has been executed using the `action description`. The states that have not been reached due to state model inference restrictions (action derivation and filters strategy, or depth limit) are not included in the table.

Based on the evaluation of the results obtained with this complex Calibre web application, we can consider the challenging functionalities that negatively affect the state model inference and change detection process, and the further improvement ideas for the analysis of the detected changes. We observe that some results share similarities with the OBS-Studio application regardless of being different systems.

6.3.1. Calibre challenging results

In the delta version from v0.6.18 to v0.6.19, the static left menu of the web application has changed with a new `Downloaded Books` item. Similar to the OBS-Studio results, this change in the static menu does not interfere with the state model inference process itself but introduces change detection *noise* in all the states of the merged model. This change detection *noise* implies that all states will be marked as changed. Then, the user needs to analyze all states to determine if there are changes or not.

Also, in the delta version from v0.6.18 to v0.6.19, we found a challenge with the **UI, Basic, and Database Configuration panels**. Due to environment resolution and visibility of web application widgets, the inference process with v0.6.18 was not able to discover 10 states that were discovered in the incremental version v0.6.19.

6.3.2. Calibre further improvements ideas based on results

In the delta version from v0.6.20 to v0.6.21, the inclusion of the `Português` option as a language is not directly visible to the user when inspecting the merged graph. It has been necessary a complementary DOM analysis to validate the existence of this change.

Table 9
OBS-Studio change detection qualitative results.

OBS-Studio	Summary of detected changes
v27.2.4 to v28.1.2	- **ALL states:** The main `Audio Mixer` and `Scene Transitions` dock panels, on the bottom side of the application, contain new icon-buttons. This provokes the change detection algorithm to detect that all states have changed. - **Help menu**: contains the new menu item `Check File Integrity` - **Crash Reports menu item**: contains a changed text from `Upload Last Crash Report` to `Upload Previous Crash Report` - **Log File menu item**: contains a changed text from `Upload Last Log File` to `Upload Previous Log File` - **Add Profile panel**: title has changed compared to the previous `Add Scene Collection` title - **Tools menu**: has removed `Declink Captions` and `Declink Output` menu items and added a new `obs-websocket Settings` item - **Declink Output panel**: has been removed - **Declink Captions panel**: has been removed - **obs-websocket Settings panel**: has been added - **Advanced Audio Properties panel**: has changed with new audio values - **Transform menu item**: includes new special shortcut texts for the `Copy Transform` and `Paste Transform` - **File menu**: has changed by removing the menu item `Always on top` - **Docks menu**: has changed the `Lock UI` and `Reset UI` menu items to `Lock Docks` and `Reset Docks` - **View menu**: has changed by adding the menu items to `Reset UI` and `Always on top`. Moreover, the menu items text has changed from `Multiview` and `Fullscreen` to `Multi-view` and `Full-screen` - **Multi-view (Windowed) panel**: has been added compared to the previously removed `Multiview (Windowed)` panel - **Settings panel**: has changed with the new option `Accessibility` - **Settings-Hotkeys panel**: has changed with a new filter by the hotkey list option `Split Recording File` - **Settings-Output panel**: has changed with a new dropdown option `Encoder Preset` - **Settings-Advanced panel**: has changed with new dropdown options `SDR White Level` and `HDR Nominal Peak Level` - **Settings-Accessibility panel**: is new in the `Settings` panel
v28.1.2 to v29.1.3	- **ALL states:** The main `Scenes` dock panel, on the bottom-left side of the application, contains a new `Open Scene Filters` icon-button. This provokes the change detection algorithm to detect that all states have changed. - **Help menu**: contains a new menu item `What is New` - **Tools menu**: has changed by renaming the menu item `obs-websocket Settings` to `WebSocket Server Settings` - **WebSocket Server Settings panel**: has been added compared to the previously removed `obs-websocket Settings` panel - **Multi-View (Full-screen) menu item**: has changed to indicate the naming of the `DISPLAY`. This affects the subsequent action transition - **Settings-General panel**: has changed due to moving the general checkbox `Automatically check for updates at startup` to a new `Updates` subpanel - **Settings-Accessibility panel**: has changed the checkbox text `Colors` to `Colours` - **Settings-Output panel**: has changed due to renaming the dropdown `Encoder` to `Video Encoder` and adding a new `Audio Encoder` dropdown. Moreover, has added an `Audio Track` option to the `Recording` subpanel - **QComboBox Collapsed** attribute has been changed in the Qt implementation. This provokes combo box items not to be displayed and clickable until the combo box is expanded
v29.1.3 to v30.0.2	- **Docks menu**: contains a new menu item `Full-Height Docks` - **Help menu**: contains two new menu items `Restart in Safe Mode` and `Release Notes` - **Restart panel**: has been added to the new SUT version - **View menu**: has changed by removing the menu item `Scene/Source List Buttons` and adding two new menu items `Scene List Mode` and `Dock Toolbars`. This affects the subsequent action transition - **Scene List Mode menu item**: has been added to the new SUT version - **Settings-Hotkeys panel**: has changed the hotkey list option `Studio Mode` into two list options `Enable Studio Mode` and `Disable Studio Mode` - **Settings-Output panel**: has indirectly changed due to moving the settings warnings float at the bottom of the page. This change is not visual in the state but exists as a widget-tree change, and it has been detected by the change detection algorithm - **Settings-Advanced panel**: has changed by adding a new `IP Family` option to the `Network` section. This change is not visual in the state but exists as a widget-tree change, and it has been detected by the change detection algorithm

As mentioned for OBS-Studio, a descriptive message indicating the changed widget could help users recognize the changes easily.

The pixelmatch technique for highlighting GUI changes has had better results than for OBS. Nonetheless, in some states with tables, the pixel comparison results do not adequately assist the user in visualizing the specific GUI changes. Highlighting the widget's position on the screen with a rectangle can help the visual comparison.

In all delta versions, the **Allowed Tags row element** action has been tracked as added and removed state instead of changed because the action description remains empty. It is necessary to improve the state model inference process to use other web widget identifiers to be included in the description of actions.

6.4. Myexpenses results

Table 11 summarizes the change detection results. The change detection algorithm for MyExpenses has been executed using the `state + action identifier`. The states that have not been reached due to state model inference restrictions (action derivation and filters strategy, or depth limit) are not included in the table.

Based on the evaluation of the results obtained with this complex MyExpenses Android application, we can consider the challenging functionalities that negatively affect the state model inference and change detection process, and the further improvement ideas for the analysis of the detected changes.

6.4.1. Myexpenses challenging results

In all versions, the hierarchical composition of some GUI elements was made up of multiple widgets. For instance, a button was composed of a clickable LinearLayout, a non-clickable RelativeLayout child, and a non-clickable TextView grandchild that contains the widget description. This causes some actions to have empty descriptions and cannot be used as action identifiers.

As observed in OBS and Calibre systems, using the `action description` for GUI change detection is a necessary approach when a delta increment affects a lot of states since it is not dependent on origin state changes. Although the `state + action identifier` is a valid approach for MyExpenses because the system versions evolve with many small delta increments, it will be beneficial to extract the description of the widget child during the model inference when a clickable Android element does not contain any description.

6.4.2. Myexpenses further improvements ideas based on results

Similar to Calibre, the pixelmatch technique for highlighting GUI changes has had better results than OBS. Nonetheless, in some states

Table 10
Calibre-Web change detection qualitative results.

Calibre-Web	Summary of detected changes
v0.6.18 to v0.6.19	- **ALL states**: The static left menu has changed with a new `Downloaded Books` item - **Edit E-mail Server Settings form**: has changed the default email `mail.example.com` to `mail.example.org` - **Tasks table**: has changed with a new `Actions` row - **Edit User admin form**: has changed the `Kindle` text to `E-Reader` - **admin's profile form**: has changed the `Kindle` text to `E-Reader` - **Add new user form**: has changed the `Kindle` text to `E-Reader` - **Edit Users table**: has changed the `Kindle` text to `E-Reader`. This change is also detected when the table elements are selected - **Users table and E-mail Server Settings panel**: has changed the `Kindle` text to `E-Reader`. Moreover, the *E-mail Server Settings panel* has changed with less text information - **Allowed Tags row element**: In the `Edit Users` table, after clicking the `Allowed Tags` row element, the Change Detection algorithm detects a state as added and removed instead of changed. This occurs because this table element does not contain an action description, and actions are not detected as corresponding actions. - **UI, Basic, and Database Configuration panels**: are included as new states in the model because the `E-mail Server Settings` panel has changed with less text information. This is indirectly provoked because the resolution and the number of visible elements affect the state model inference process
v0.6.19 to v0.6.20	- **Add New User form**: has capitalized the page title, has changed the `E-Reader` text to `eReader`, and has changed the `E-mail` text to `Email` - **admin's Profile form**: has capitalized the page title, has changed the `E-Reader` text to `eReader`, has changed the `E-mail` text to `Email`, and has added `admin@example.org` as default email - **Edit User admin form**: has changed the `E-Reader` text to `eReader`, has changed the `E-mail` text to `Email`, and has added `admin@example.org` as default email - **Edit Users table**: has changed the `E-Reader` text to `eReader`, has changed the `E-mail` text to `Email address`, and has added `admin@example.org` as default email. This change is also detected when the table elements are selected. Moreover, the new email `admin@example.org` is now clickable and editable in a new SUT state - **Users table and Email Server Settings panel**: has changed the `E-Reader` text to `eReader`, has changed the `E-mail` text to `Email`, and has added `admin@example.org` as default email - **Default Visibilities for New Users configuration**: has changed by capitalizing the checkbox text options - **Default Settings for New Users configuration**: has changed by capitalizing the checkbox text options. This change is not visual in the state but exists at the DOM level - **Basic Configuration panel**: has changed with a new "Securitiy settings" option. This change is detected in 4 subsequent states - **Securitiy settings option panel**: has been added to the new SUT version - **Edit Email Server Settings form**: has changed the `E-mail` text to `Email`. Due to the change in the action description, the state is detected as added and removed instead of changed. - **Allowed Tags row element**: In the `Edit Users` table, after clicking the `Allowed Tags` row element, the Change Detection algorithm detects a state as added and removed instead of changed. This occurs because this table element does not contain an action description, and actions are not detected as corresponding actions.
v0.6.20 to v0.6.21	- **Basic Configuration panel**: has fixed a typo text from `Securitiy settings` to `Security settings`. This change is detected in 4 subsequent states. This GUI typo issue was reported and fixed by the project contributors[a] - **Security settings option panel**: is detected as added and removed due to the change in the action description - **LogFile Configuration dropdown**: adds the new default value `/config/access.log` for the logfile location - **Feature Configuration dropdown**: changes the checkbox text option `Please ensure that users also have upload permissions` - **Edit User admin form**: has changed by updating the dropdown `Language` to include the `Português` option. This change is not visual in the state but exists at the DOM level - **admin's Profile form**: has changed by updating the dropdown `Language` to include the `Português` option. This change is not visual in the state but exists at the DOM level - **Add New User form**: has changed by updating the dropdown `Language` to include the `Português` option. This change is not visual in the state but exists at the DOM level - **Default Settings for New Users configuration**: has changed by updating the dropdown `Default Language` to include the `Português` option. This change is not visual in the state but exists at the DOM level - **Edit Users table**: has changed by updating the select `Language` to include the `Português` option. This change is not visual in the state but exists at the DOM level. This change is also detected when the table elements are selected - **Users table page**: contains a non-visible widget that contains a system version change. This change exists and has been detected at the DOM level - **Allowed Tags row element**: In the `Edit Users` table, after clicking the `Allowed Tags` row element, the Change Detection algorithm detects a state as added and removed instead of changed. This occurs because this table element does not contain an action description, and actions are not detected as corresponding actions

[a] https://github.com/janeczku/calibre-web/issues/2811.

with a lot of text, the pixel comparison results do not adequately assist the user in visualizing the specific GUI changes.

6.5. Answer to the research question

In order to answer RQ2: *Does the ChangeDetection tool detect delta GUI changes when using inferred state models?*, we conducted a manual qualitative evaluation of detected GUI delta changes with the OBS-Studio, Calibre-Web, and MyExpenses applications. Our results have demonstrated positive GUI change detection results and the potential to uncover GUI failures. Consequently, we reject H_0: *The ChangeDetection tool does not detect delta GUI changes using inferred state models.*

Our proposed approach of using inferred state models addresses existing state-of-the-art limitations in GUI change detection to detect and highlight GUI change transitions for a diverse range of desktop, web, and mobile systems. Moreover, our tool provides an interactive interface with a merged model that users can easily use to visually analyze GUI changes. These ideas could help to pave the way to a delta GUI change detection standard that streamlines the efforts of developers, testers, and other project contributors in identifying functionalities that have been removed, added, or modified.

Nonetheless, as a result of novel research, our findings with complex GUI applications also expose challenges we will need to address in the future to improve the delta GUI change detection technique using inferred state models.

Table 11
MyExpenses change detection qualitative results.

MyExpenses	Summary of detected changes
v3.6.6 to v3.6.7	- **Settings Data panel**: has changed by adding a new `Unmapped Transactions` option - **Parties panel**: has changed by removing the `merge` option that should only be shown if there are at least two parties - **Select Category panel**: has changed with two new `Expense` and `Income` filter options - **New Main Category panel**: has changed with two new `Expense` and `Income` filter options - **Select Category-Help panel**: has changed with a new `Expense/Income FAQ` option - **Discard confirmation panel**: has been detected as added when the user does not complete the creation of a new transaction template. Although there exist code changes that modify the back behavior and discard panel, we consider this detected change a false positive because we were not able to reproduce this functionality
v3.6.7 to v3.6.8	- **Premium feature - Split transaction panel**: has changed with a new radio button option - **Premium feature - FinTS panel**: has changed with a new radio button option - **Premium feature - Budgeting panel**: has changed with a new radio button option - **Premium feature - Scan receipt panel**: has changed with a new radio button option - **Backup folder dialog**: has changed because it shows a Java object string instead of the backup folder directory. We reported this bug in the project repository, which was resolved by the project developer[a] - **False positive - Typing Date**: The dynamic date text detects a change due to an inadequate abstraction that does not match the additional `after` text
v3.6.8 to v3.6.9	- **Categories panel**: has changed the button `Setup default categories` to `Transfer` - **Select Category panel**: has changed the button `Setup default categories` to `Transfer` - **Select Category unselected checkbox**: now transits to a newly added state with the `Transfer` button - **Backup folder dialog**: continues showing another Java object string instead of the backup folder directory

[a] https://github.com/mtotschnig/MyExpenses/issues/1378.

7. Threats to validity

This section mentions some threats that could affect the validity of our study [70,75,76].

- **Construct validity (Systematic mapping):** The decision on exclusion and inclusion criteria during the systematic mapping of the literature may be unintentionally biased by the researchers' knowledge and judgment. This is mainly because some techniques reported in studies that aim at repairing test scripts or detecting cross-browser and cross-device differences are strongly connected to the delta GUI change detection topic. Moreover, in other studies, it is not clear how changes are reported with textual or visual information. To mitigate this threat, we have documented and shared the process results, as well as reported other interesting related work studies.
- **Construct validity (Change Detection tool):** The state model inference was limited to 3 click-action lengths to control the number of discovered states and allow the inference of a partial but complete state model. However, this causes certain internal states of the application to not be inferred in the model, and the change detection approach will not be able to detect and highlight changes in these states.
- **Content validity (ChangeDetection tool):** The qualitative evaluation to determine the accuracy of the detected GUI changes was performed manually by inspecting the diverse SUT versions from the open-source software repositories. Although the results are described in a complete document, human error is possible in determining these results.
- **Internal validity (Systematic mapping):** The results of the systematic mapping of the literature search query may not retrieve all the relevant research works related to the GUI change detection topic. To mitigate this threat, we used Scopus, the largest database of peer-reviewed scientific literature, and we validated the employed terminology with a small set of relevant works found during our previous research [12].
- **Internal validity (ChangeDetection tool):** A unique researcher, with experience in the TESTAR state model inference, configured the abstraction strategy of the SUT objects. This introduces subjectivity and potential bias based on their tacit knowledge and decisions. Different researchers may consider that using a different abstraction strategy is more adequate.
- **External validity (ChangeDetection tool):** We use one open-source Desktop, Web, and Android application to conduct the study. We have demonstrated the potential of integrating the change detection approach to detect and highlight GUI changes between delta versions. Moreover, we mentioned diverse inference and abstraction challenges that affect this delta GUI change detection approach. Nevertheless, it is necessary to extend the research with diverse types of applications to be able to generalize the results.
  As an additional external validity threat, all the state models used in this study are inferred using the TESTAR tool. For this reason, it is necessary to investigate other state models of event flow graph tools to extend and improve the ChangeDetection tool.

8. Discussion

In today's software projects, where development methodologies emphasize rapid and iterative updates with frequent delta increments, it is crucial to integrate techniques that can automatically detect and verify the correctness of software changes. These techniques are essential to support developers and testers by providing them with analytics and visual results. While code review has emerged and been embraced as a key strategy to ensure readability, consistency, and correctness in all software projects, our systematic mapping of the literature indicates that despite the existence of state-of-the-art delta GUI change detection techniques, their ideas and implementations remain scattered depending on the type of system or are not used for the main purpose of detecting and highlighting changes. We provide a new delta GUI change detection approach using inferred models to tackle this limitation.

This study presents how the automated inference of GUI models, in conjunction with change detection algorithms for model transitions, provides an interactive model that automatically highlights delta GUI changes. This interactive model can be useful for developers, testers, and other project contributors to detect added, removed, and changed functionalities. Furthermore, the empirical evaluation demonstrates the versatility of the state model inference technique, showcasing its applicability to a diverse range of systems like desktop, web, and mobile applications.

Analyzing real and complex open-source systems reveals both the benefits and challenges of employing inferred models for delta GUI change detection. The technique automatically identifies and highlights GUI states changed between versions, which can support developers and testers in validating changes and uncovering potential issues. Notable examples include the detected Calibre web state that contains a typo issue that was already reported and fixed by their community and the successful detection of a Java object display issue reported by our team to the MyExpenses mobile software project.

Nevertheless, the complexity of GUI systems, characterized by a multitude of states with dynamic behaviors, poses significant challenges that require extensive research and knowledge in inference and abstraction strategies when inferring state models. An inadequate inference or abstraction strategy can lead to erroneous change detection due to a lack of inferred GUI transitions if the model is not partially complete or due to dynamic alterations and nondeterministic behaviors. It is worth mentioning that the implementation of a Domain Specific Language (DSL) that allows the configurable level of abstraction and inference strategies could help to deal with the intrinsic state explosion, dynamism, and non-determinism challenges of complex GUI systems that will remain.

Other challenges, such as the model traverse or the existence of *noise*, arise when comparing these models to detect changes. Introducing the capability for users to configure and utilize action descriptions during model comparison has proven to enhance the traversal of models by facilitating the comparison of subsequent transitions. Hence, in further implementations, integrating configuration functions or DSL instructions that allow users to ignore specific widgets (e.g., the static widgets from the OBS panel or the Calibre menu) could mitigate the propagation of change detection *noise* across the states of the merged model.

When it comes to empirically evaluating the accuracy and inaccuracy of the delta GUI change detection technique, it becomes crucial to extend the assessment by incorporating metrics for false and true positives and negatives, accompanied by statistical analysis. However, conducting such evaluations demands a substantial understanding of the SUT to correlate the GUI with the underlying classes and code. Moreover, in certain systems like OBS desktop and Calibre web, where there are numerous commits and code changes between release versions, the manual inspection becomes impractical due to the sheer volume of changes. For this reason, the empirical evaluation review for these systems should involve a developer or tester expert in the SUT, or this evaluation should be performed on smaller SUT delta commit increments.

Additionally, obtaining feedback on the perception of the effectiveness, efficiency, usability, and challenges of our proposal is of significant importance. Under the European Innovation Alliance for Testing Education (ENACTEST) project, which aims to identify and design seamless teaching materials for testing that align with industry and learning needs [77], we conducted a preliminary evaluation of our change detection approach and the interactive web interface. This experimental evaluation involved six master's students enrolled in a software testing course.

The experiment simulated a pull request with several commits representing a new delta version of a web SUT. The students individually verified whether changes at the code level resulted in changes at the GUI level using the change detection tool. They were asked to determine if the changes indicated by the tool were correct GUI changes or if the commit changes introduced GUI failures. Usability evaluation methods included a cognitive walkthrough to understand the tool's learnability for new users, Likert-scale questionnaires to gather qualitative and quantitative information about the user experience, and a final group interview to capture the students' experiences and impressions [78–81].

Feedback from this preliminary evaluation indicates that the change detection approach can help project newcomers, such as developers, testers, or other collaborators, to visualize and understand the software. It can reveal how code and GUI are related when the change detection models are reviewed during delta pull requests. Additionally, this evaluation results also pointed to technical aspects for visual improvements, like the need to highlight changes at the DOM level, even if the GUI visuals remain the same.

This feedback motivates us to continue improving the change detection approach and the interactive web interface. For a broader generalization of the cognitive and usability results, we plan to continue the evaluation under the ENACTEST project. We expect to gather feedback from industrial partners and conduct further experiment evaluations with bachelor's and master's students. Future experiments aim to evaluate individual and collaborative cognitive and usability perceptions of the change detection tool. Following ENACTEST principles, we will conduct formal experiments by recording and analyzing videos of user actions [82], studying the constructive interaction of two users discovering the tool's characteristics together [83], continuing with individual and group interviews, and formalizing the results by analyzing and publishing the evaluation outcomes.

We advocate this approach can provide the beginning of a path toward standardizing the use of inferred models to help detect GUI changes across diverse systems encompassing desktop, web, and mobile applications commonly used by most users in their daily activities. Adopting this practice will streamline the work of the developers, testers, and other stakeholders involved in software development projects, improving quality standards and enhancing user software usage experiences. As with code-level change detection and reporting practices, this approach requires future research, dealing with challenges and decisions that will increasingly standardize this practice in software system projects. Since the developed tool and replication data are open-source, we hope other researchers continue with future empirical studies or compare our tool with other GUI change detection techniques.

9. Conclusions and future work

This paper presents a delta GUI change detection approach that utilizes automated inferred state models to identify and highlight differences in the GUI between different SUT versions. Through a literature review process, we have identified a gap in existing techniques or tools capable of automatically detecting and highlighting delta GUI changes for a diverse range of applications in desktop, web, and mobile systems. Furthermore, our examination indicates that outside web systems, delta GUI change detection techniques are mainly researched in regression testing studies that lack emphasis on reporting textual or visual GUI changes to end-users.

Takeaway: *In the state-of-the-art, there is a gap in existing techniques or tools capable of automatically detecting and highlighting delta GUI changes for a diverse range of desktop, web, and mobile applications, regardless of the internal architecture of the software.*

Recognizing the significance of integrating a delta GUI change detection technique into software repositories for software quality assurance, we provide a novel open-source tool to improve a GUI change detection approach that can detect and highlight GUI changes from 3 different software systems. The GUI change detection approach recursively compares two inferred state models from distinct software versions to mark the states and actions that have been changed, added, or removed. Subsequently, a merged graph technique allows the visualization of these changes in an interactive web interface. To substantiate our approach and contribute valuable insights into the software testing field, we have conducted an empirical evaluation using representative SUTs from real and widely adopted applications.

Takeaway: *A novel delta change detection open-source tool that uses inferred GUI models from an automated scriptless testing tool can automatically detect and offer the visualization of GUI changes.*

The empirical evaluation results demonstrate the feasibility of employing automatic inferred GUI state models for executing an algorithm that detects GUI changes, complemented by a merge graph technique that highlights these changes to end-users. These techniques are developed in a dedicated ChangeDetection tool, designed to operate independently with respect to the specific SUT technical requirements. Nevertheless, it is important to note that, similar to any technique that relies on model inference, the efficacy of the ChangeDetection tool depends on the design of adequate inference and abstraction strategies.

> **Takeaway:** *An empirical evaluation, performed with real and widely adopted GUI applications, demonstrates the feasibility of using the change detection approach for detecting and highlighting GUI changes.*

> **Takeaway:** *The efficacy of the GUI change detection approach depends on the design of adequate inference and abstraction strategies.*

Finally, we conducted a preliminary experiment with master's students to obtain cognitive and usability feedback on how the change detection approach can help newcomers visualize and understand a software project. Additionally, this experiment also pointed to technical aspects for future visual improvement.

> **Takeaway:** *The change detection approach can help newcomers visualize and understand the changes in different versions of a software project.*

Future work will encompass several objectives aimed at enhancing the efficacy and usability of our GUI change detection tool. The inclusion of configurable abstraction identifiers for state model actions (see 4.3) has proven to enhance the detection and highlighting of GUI changes even if the origin state identifier has changed. For this reason, we plan to study the feasibility of introducing new configurable abstraction options that allow ignoring certain widgets from the widget tree, such as static menu or panel widgets, to prevent the propagation of change detection *noise*.

For web applications, we detected that the change detection algorithm effectively detects DOM changes of existing elements that are not visible in the GUI. Future work aims to extend the visualization of the changed state with a textual description that summarizes these changes. For example, integrating Artificial Intelligence (AI) techniques such as Natural Language (NL) comments generation [84] adapted to DOM properties.

In terms of experimentation, we will delve into change detection research with state models that rely on typing and scroll actions, addressing more complex user GUI interactions. These interactions are essential to our future planning of using scriptless model inference and change detection with software from industry partners. Moreover, technical enhancements will aim to optimize memory management for the tool when comparing large models, ensuring efficient performance even in resource-intensive scenarios.

For future evaluation, we plan to continue validating the cognitive and usability aspects of the change detection approach and the web interface tool in collaboration with industry and academic partners of the ENACTEST project, as well as other interested stakeholders. These evaluations will help determine the preliminary feedback on whether the GUI change detection approach helps to overcome the technical knowledge challenges faced by software project newcomers [85]. Additional evaluation, improvements, and documentation of this open-source change detection software project will require considering regional diversity to empower worldwide contributions [86]. Furthermore, to continue the way to standardize this technique, we plan to conduct interviews with experts in the GUI testing field and developers and testers involved in complex GUI systems to obtain the perception of effectiveness, efficiency, and challenges of our proposal.

CRediT authorship contribution statement

Fernando Pastor Ricós: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Beatriz Marín:** Writing – review & editing, Writing – original draft, Validation, Methodology, Investigation, Conceptualization. **Tanja E.J. Vos:** Writing – review & editing, Writing – original draft, Methodology, Investigation, Conceptualization. **Rick Neeft:** Software, Methodology, Investigation, Conceptualization. **Pekka Aho:** Software, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Fernando Pastor Ricos reports financial support was provided by Erasmus. Beatriz Marin reports financial support was provided by Erasmus. Tanja E.J. Vos reports financial support was provided by Erasmus. If there are other authors they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The research article contains references to the software code and evaluation data.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author(s) used ChatGPT and Grammarly in order to improve the grammar and syntax of the content. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

Acknowledgment

This research has been funded by the project European Innovation Alliance for Testing Education (ENACTEST), ERASMUS+ number 101055874,2022–2025.

References

[1] J. Pantiuchina, M. Mondini, D. Khanna, X. Wang, P. Abrahamsson, Are software startups applying agile practices? The state of the practice from a large survey, in: Agile Processes in Software Engineering and Extreme Programming: 18th International Conference, XP 2017, Cologne, Germany, May 22-26, 2017, Proceedings 18, Springer International Publishing, 2017, pp. 167–183.
[2] R. Hoda, N. Salleh, J. Grundy, The rise and evolution of agile software development, IEEE Softw. 35 (5) (2018) 58–63.
[3] G. Giachetti, J.L. de la Vara, B. Marín, A model-driven approach to adopt good practices for agile process configuration and certification, Comput. Stand. Interfaces 86 (2023) 103737.
[4] S. Al-Saqqa, S. Sawalha, H. AbdelNabi, Agile software development: Methodologies and trends, Int. J. Interact. Mob. Technol. 14 (11) (2020).
[5] C. Vassallo, F. Palomba, A. Bacchelli, H.C. Gall, Continuous code quality: are we (really) doing that? in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018, pp. 790–795.
[6] M. Saadatmand, E.P. Enoiu, H. Schlingloff, M. Felderer, W. Afzal, Smartdelta: automated quality assurance and optimization in incremental industrial software systems development, in: 2022 25th Euromicro Conference on Digital System Design, DSD, IEEE, 2022, pp. 754–760.
[7] H.M. Idrus, et al., Tacit knowledge in software testing: A systematic review, in: 2019 6th International Conference on Research and Innovation in Information Systems, ICRIIS, IEEE, 2019, pp. 1–6.
[8] A. Bons, B. Marín, P. Aho, T.E. Vos, Scripted and scriptless GUI testing for web applications: An industrial case, Inf. Softw. Technol. 158 (2023) 107172.
[9] C. Sadowski, E. Söderberg, L. Church, M. Sipko, A. Bacchelli, Modern code review: a case study at google, in: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, 2018, pp. 181–190.

[10] J. Santos, D. Alencar da Costa, U. Kulesza, Investigating the impact of continuous integration practices on the productivity and quality of open-source projects, in: Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2022, pp. 137–147.

[11] Y. Yu, H. Wang, G. Yin, T. Wang, Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment? Inf. Softw. Technol. 74 (2016) 204–218.

[12] F.P. Ricós, R. Neeft, B. Marín, T.E. Vos, P. Aho, Using GUI change detection for delta testing, in: International Conference on Research Challenges in Information Science, Springer, 2023, pp. 509–517.

[13] B. Kitchenham, S. Charters, et al., Guidelines for performing systematic literature reviews in software engineering, 2007.

[14] B.A. Kitchenham, D. Budgen, O.P. Brereton, Using mapping studies as the basis for further research–a participant-observer case study, Inf. Softw. Technol. 53 (6) (2011) 638–651.

[15] O. Rodríguez-Valdés, T.E. Vos, P. Aho, B. Marín, 30 Years of automated GUI testing: A bibliometric analysis, in: Quality of Information and Communications Technology: 14th International Conference, QUATIC 2021, Algarve, Portugal, September 8–11, 2021, Proceedings 14, Springer, 2021, pp. 473–488.

[16] R. Pranckutė, Web of Science (WoS) and Scopus: The titans of bibliographic information in today's academic world, Publications 9 (1) (2021) 12.

[17] M. Bures, Change detection system for the maintenance of automated testing, in: Testing Software and Systems: 26th IFIP WG 6.1 International Conference, ICTSS 2014, Madrid, Spain, September 23-25, 2014. Proceedings 26, Springer, 2014, pp. 192–197.

[18] C. Wohlin, Guidelines for snowballing in systematic literature studies and a replication in software engineering, in: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, 2014, pp. 1–10.

[19] M. Grechanik, C.W. Mao, A. Baisal, D. Rosenblum, B.M. Hossain, Differencing graphical user interfaces, in: 2018 IEEE International Conference on Software Quality, Reliability and Security, QRS, IEEE, 2018, pp. 203–214.

[20] S. Raina, A.P. Agarwal, An automated tool for regression testing in web applications, ACM SIGSOFT Softw. Eng. Notes 38 (4) (2013) 1–4.

[21] T.A. Walsh, G.M. Kapfhammer, P. McMinn, Automatically identifying potential regressions in the layout of responsive web pages, Softw. Test. Verif. Reliab. 30 (6) (2020) e1748.

[22] D. Roest, A. Mesbah, A. Van Deursen, Regression testing ajax applications: Coping with dynamism, in: 2010 Third International Conference on Software Testing, Verification and Validation, IEEE, 2010, pp. 127–136.

[23] Z. Gao, C. Fang, A.M. Memon, Pushing the limits on automation in GUI regression testing, in: 2015 IEEE 26th International Symposium on Software Reliability Engineering, ISSRE, IEEE, 2015, pp. 565–575.

[24] H. Tanno, Y. Adachi, Y. Yoshimura, K. Natsukawa, H. Iwasaki, Region-based detection of essential differences in image-based visual regression testing, J. Inf. Process. 28 (2020) 268–278.

[25] Y. Adachi, H. Tanno, Y. Yoshimura, A method to mask Dynamic Content Areas based on positional relationship of screen elements for visual regression testing, in: 2020 IEEE 44th Annual Computers, Software, and Applications Conference, COMPSAC, IEEE, 2020, pp. 1755–1760.

[26] Y. Xiong, M. Xu, T. Su, J. Sun, J. Wang, H. Wen, G. Pu, J. He, Z. Su, An empirical study of functional bugs in android apps, in: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, 2023, pp. 1319–1331.

[27] M. Li, J. Wang, L. Damata, TAO project: An intuitive application UI test toolset, in: 2009 Sixth International Conference on Information Technology: New Generations, IEEE, 2009, pp. 796–800.

[28] K. Moran, C. Watson, J. Hoskins, G. Purnell, 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.

[29] S. Flesca, F. Furfaro, E. Masciari, Monitoring web information changes, in: Proceedings International Conference on Information Technology: Coding and Computing, IEEE, 2001, pp. 421–425.

[30] M. Qiang, S. Miyazaki, K. Tanaka, WebSCAN: Discovering and notifying important changes of web sites, in: Database and Expert Systems Applications: 12th International Conference, DEXA 2001 Munich, Germany, September 3–5, 2001 Proceedings 12, Springer, 2001, pp. 587–598.

[31] L. Francisco-Revilla, F.M. Shipman III, R. Furuta, U. Karadkar, A. Arora, Perception of content, structure, and presentation changes in web-based hypertext, in: Proceedings of the 12th ACM Conference on Hypertext and Hypermedia, 2001, pp. 205–214.

[32] L. Francisco-Revilla, F. Shipman, R. Furuta, U. Karadkar, A. Arora, Managing change on the web, in: Proceedings of the 1st ACM/IEEE-CS Joint Conference on Digital Libraries, 2001, pp. 67–76.

[33] L. Liu, W. Tang, D. Buttler, C. Pu, Information monitoring on the web: a scalable solution, World Wide Web 5 (2002) 263–304.

[34] N. Agrawal, R. Ananthanarayanan, R. Gupta, S. Joshi, R. Krishnapuram, S. Negi, The eshopmonitor: A comprehensive data extraction tool for monitoring web sites, IBM J. Res. Dev. 48 (5.6) (2004) 679–692.

[35] Z. Dalal, S. Dash, P. Dave, L. Francisco-Revilla, R. Furuta, U. Karadkar, F. Shipman, Managing distributed collections: evaluating web page changes, movement, and replacement, in: Proceedings of the 4th ACM/IEEE-CS Joint Conference on Digital Libraries, 2004, pp. 160–168.

[36] J. Jacob, A. Sachde, S. Chakravarthy, CX-DIFF: a change detection algorithm for XML content and change visualization for WebVigiL, Data Knowl. Eng. 52 (2) (2005) 209–230.

[37] J. Teevan, S.T. Dumais, D.J. Liebling, R.L. Hughes, Changing how people view changes on the web, in: Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology, 2009, pp. 237–246.

[38] A. Jatowt, Y. Kawai, K. Tanaka, Browsing assistant for changing pages, in: Intelligent Agents in the Evolution of Web and Applications, Springer, 2009, pp. 137–160.

[39] H. Khandagale, P. Halkarnikar, A novel approach for web page change detection system, Int. J. Comput. Theory Eng. 2 (3) (2010) 364.

[40] Z. Pehlivan, M. Ben-Saad, S. Gançarski, Vi-DIFF: Understanding web pages changes, in: Database and Expert Systems Applications: 21st International Conference, DEXA 2010, Bilbao, Spain, August 30-September 3, 2010, Proceedings, Part I 21, Springer, 2010, pp. 1–15.

[41] F. Shao, R. Xu, W. Haque, J. Xu, Y. Zhang, W. Yang, Y. Ye, X. Xiao, WebEvo: taming web application evolution via detecting semantic structure changes, in: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2021, pp. 16–28.

[42] M. Grechanik, Q. Xie, C. Fu, Maintaining and evolving GUI-directed test scripts, in: 2009 IEEE 31st International Conference on Software Engineering, IEEE, 2009, pp. 408–418.

[43] S. Zhang, H. Lü, M.D. Ernst, Automatically repairing broken workflows for evolving gui applications, in: Proceedings of the 2013 International Symposium on Software Testing and Analysis, 2013, pp. 45–55.

[44] Z. Gao, Z. Chen, Y. Zou, A.M. Memon, Sitar: Gui test script repair, Ieee Trans. Softw. Eng. 42 (2) (2015) 170–186.

[45] S.R. Choudhary, D. Zhao, H. Versee, A. Orso, Water: Web application test repair, in: Proceedings of the First International Workshop on End-To-End Test Script Engineering, 2011, pp. 24–29.

[46] M. Hammoudi, G. Rothermel, A. Stocco, Waterfall: An incremental approach for repairing record-replay tests of web applications, in: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2016, pp. 751–762.

[47] A. Stocco, R. Yandrapally, A. Mesbah, Visual web test repair, in: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018, pp. 503–514.

[48] H. Kirinuki, H. Tanno, K. Natsukawa, COLOR: correct locator recommender for broken test scripts using various clues in web application, in: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering, SANER, IEEE, 2019, pp. 310–320.

[49] M. Nass, E. Alégroth, R. Feldt, M. Leotta, F. Ricca, Similarity-based web element localization for robust test automation, ACM Trans. Softw. Eng. Methodol. 32 (3) (2023) 1–30.

[50] X. Li, N. Chang, Y. Wang, H. Huang, Y. Pei, L. Wang, X. Li, ATOM: Automatic maintenance of GUI test scripts for evolving mobile applications, in: 2017 IEEE International Conference on Software Testing, Verification and Validation, ICST, IEEE, 2017, pp. 161–171.

[51] F. Song, Z. Xu, F. Xu, An xpath-based approach to reusing test scripts for android applications, in: 2017 14th Web Information Systems and Applications Conference, WISA, IEEE, 2017, pp. 143–148.

[52] N. Chang, L. Wang, Y. Pei, S.K. Mondal, X. Li, Change-based test script maintenance for android apps, in: 2018 Ieee International Conference on Software Quality, Reliability and Security (Qrs), IEEE, 2018, pp. 215–225.

[53] M. Pan, T. Xu, Y. Pei, Z. Li, T. Zhang, X. Li, Gui-guided test script repair for mobile apps, IEEE Trans. Softw. Eng. 48 (3) (2020) 910–929.

[54] T. Xu, M. Pan, Y. Pei, G. Li, X. Zeng, T. Zhang, Y. Deng, X. Li, Guider: Gui structure and vision co-guided test script repair for android apps, in: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, 2021, pp. 191–203.

[55] A. Mesbah, M.R. Prasad, Automated cross-browser compatibility testing, in: Proceedings of the 33rd International Conference on Software Engineering, 2011, pp. 561–570.

[56] H. Tanaka, X-BROT: prototyping of compatibility testing tool for web application based on document analysis technology, in: 2019 International Conference on Document Analysis and Recognition Workshops, ICDARW, 7, IEEE, 2019, pp. 18–21.

[57] H. Tanaka, Automating web GUI compatibility testing using X-BROT: Prototyping and field trial, in: Document Analysis and Recognition–ICDAR 2021 Workshops: Lausanne, Switzerland, September 5–10, 2021, Proceedings, Part II 16, Springer, 2021, pp. 255–267.

[58] Y. Ren, Y. Gu, Z. Ma, H. Zhu, F. Yin, Cross-device difference detector for mobile application GUI compatibility testing, in: 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICSTW, IEEE, 2022, pp. 253–260.

[59] T.E. Vos, P. Aho, F. Pastor Ricos, O. Rodriguez-Valdes, A. Mulders, Testar–scriptless testing through graphical user interface, Softw. Test. Verif. Reliab. 31 (3) (2021) e1771.
[60] A. Mulders, O.R. Valdes, F.P. Ricós, P. Aho, B. Marín, T.E. Vos, State model inference through the GUI using run-time test generation, in: International Conference on Research Challenges in Information Science, Springer, 2022, pp. 546–563.
[61] F.P. Ricós, A. Slomp, B. Marín, P. Aho, T.E. Vos, Distributed state model inference for scriptless GUI testing, J. Syst. Softw. 200 (2023) 111645.
[62] T. Jansen, F.P. Ricós, Y. Luo, K. van der Vlist, R. van Dalen, P. Aho, T.E. Vos, Scriptless GUI testing on mobile applications, in: 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security, QRS, IEEE, 2022, pp. 1103–1112.
[63] I. Prasetya, F. Pastor Ricós, F.M. Kifetew, D. Prandi, S. Shirzadehhajimahmood, T.E. Vos, P. Paska, K. Hovorka, R. Ferdous, A. Susi, et al., An agent-based approach to automated game testing: an experience report, in: Proceedings of the 13th International Workshop on Automating Test Case Design, Selection and Evaluation, 2022, pp. 1–8.
[64] F.P. Ricós, B. Marín, T. Vos, J. Davidson, K. Hovorka, Scriptless testing for an industrial 3D sandbox game, in: International Conference on Evaluation of Novel Approaches To Software Engineering, ENASE-Proceedings, Vol. 1, SciTePress, 2024, pp. 51–62.
[65] F.P. Ricós, P. Aho, T. Vos, I.T. Boigues, E.C. Blasco, H.M. Martínez, Deploying TESTAR to enable remote testing in an industrial CI pipeline: a case-based evaluation, in: Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles: 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20–30, 2020, Proceedings, Part I 9, Springer, 2020, pp. 543–557.
[66] M. Nass, E. Alégroth, R. Feldt, Why many challenges with GUI test automation (will) remain, Inf. Softw. Technol. 138 (2021) 106625.
[67] W. Wang, S. Sampath, Y. Lei, R. Kacker, R. Kuhn, J. Lawrence, Using combinatorial testing to build navigation graphs for dynamic web applications, Softw. Test. Verif. Reliab. 26 (4) (2016) 318–346.
[68] X.-F. Qi, Y.-L. Hua, P. Wang, Z.-Y. Wang, Leveraging keyword-guided exploration to build test models for web applications, Inf. Softw. Technol. 111 (2019) 110–119.
[69] K. Andrews, M. Wohlfahrt, G. Wurzinger, Visual graph comparison, in: 13th International Conference Information Visualisation, IEEE, 2009, pp. 62–67.
[70] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer Science & Business Media, 2012.
[71] T.E. Vos, B. Marín, M.J. Escalona, A. Marchetto, A methodological framework for evaluating software testing techniques and tools, in: 2012 12th International Conference on Quality Software, IEEE, 2012, pp. 230–239.
[72] OBS-Project, Free and open source software for live streaming and screen recording, 2012, 2023, Last accessed: 9 Dec 2022, URL https://github.com/obsproject/obs-studio.
[73] Calibre, Web app for browsing, reading and downloading ebooks, 2006, 2023, Last accessed: 9 Dec 2022, URL https://github.com/janeczku/calibre-web.
[74] M. Totschnig, Android expenses tracking app, 2012, 2023, Last accessed: 9 Dec 2022, URL https://github.com/mtotschnig/MyExpenses.
[75] P. Ralph, E. Tempero, Construct validity in software engineering research and software metrics, in: 22nd EASE, ACM, 2018, pp. 13–23.
[76] R. Feldt, A. Magazinius, Validity threats in empirical software engineering research-an initial survey, in: Seke, 2010, pp. 374–379.
[77] B. Marín, T.E. Vos, A.C. Paiva, A.R. Fasolino, M. Snoeck, et al., ENACTEST-European innovation alliance for testing education, in: Proceedings of RCIS 2022 Workshops and Research Projects Track Co-Located with the 16th International Conference on Research Challenges in Information Science, 2022.
[78] J. Nielsen, Usability Engineering, Morgan Kaufmann, 1994.
[79] J. Rieman, M. Franzke, D. Redmiles, Usability evaluation with the cognitive walkthrough, in: Conference Companion on Human Factors in Computing Systems, 1995, pp. 387–388.
[80] A. Solano, C.A. Collazos, C. Rusu, H.M. Fardoun, Combinations of methods for collaborative evaluation of the usability of interactive software systems, Adv. Hum.-Comput. Interact. 2016 (1) (2016) 4089520.
[81] N. Doorn, T.E. Vos, B. Marín, Towards understanding students' sensemaking of test case design, Data Knowl. Eng. 146 (2023) 102199.
[82] F. Cammaerts, M. Snoeck, A.C. Paiva, Collecting cognitive strategies applied by students during test case design, in: Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, 2023, pp. 455–459.
[83] C. O'Malley, S. Draper, M. Riley, Constructive interaction: a method for studying user-computer-user interaction, in: IFIP INTERACT'84 First International Conference on Human-Computer Interaction, 1984.
[84] S. Panthaplackel, P. Nie, M. Gligoric, J.J. Li, R. Mooney, Learning to update natural language comments based on code changes, in: D. Jurafsky, J. Chai, N. Schluter, J. Tetreault (Eds.), Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, Online, 2020, pp. 1853–1868, http://dx.doi.org/10.18653/v1/2020.acl-main.168, URL https://aclanthology.org/2020.acl-main.168.
[85] I. Steinmacher, T.U. Conte, C. Treude, M.A. Gerosa, Overcoming open source project entry barriers with a portal for newcomers, in: Proceedings of the 38th International Conference on Software Engineering, 2016, pp. 273–284.
[86] G.A.A. Prana, D. Ford, A. Rastogi, D. Lo, R. Purandare, N. Nagappan, Including everyone, everywhere: Understanding opportunities and challenges of geographic gender-inclusion in oss, IEEE Trans. Softw. Eng. 48 (9) (2021) 3394–3409.