Can Videos as a By-Product of GUI Testing Help Developers Understand GUI Tests?

Jianwei Shi*, Oliver Karras†, Martin Obaidi*, and Malvin Tandun*
*Leibniz Universität Hannover, Software Engineering Group, Hannover, Germany
Email: {jianwei.shi, martin.obaidi}@inf.uni-hannover.de, malvin.tandun@stud.uni-hannover.de
†TIB - Leibniz Information Centre for Science and Technology, Hannover, Germany
Email: oliver.karras@tib.eu

***Abstract*—[Context] Software with graphical user interfaces (GUIs) is used in everyday life. Users expect working software when they interact with a software product. For this reason, testing and defect corrections are required. However, understanding GUI tests is challenging, as the generated textual test reports lack the dynamic details about interaction steps to reproduce a GUI defect. [Objective] We want to revise the practice of GUI testing and use videos as a by-product to capture and convey these dynamic details. [Method] Based on the video as a by-product approach, we propose to generate videos as a by-product of GUI testing, integrating annotations and test outputs directly into the videos. These videos serve as supplementary material for documenting dynamic test executions in addition to static test reports (e.g., texts, screenshots). In an experiment, we asked 11 participants to distinguish between true and false defects in watching videos and screenshots of four test cases. We also conducted structured interviews to ask the participants about their opinions for these two documentation options. [Results & Conclusion] Our results show visible differences between the video and the screenshots in effectiveness and efficiency in defect analysis, but cannot prove the differences statistically. The listed advantages and disadvantages of both documentations are complemental. Both documentations are helpful and thus videos as a by-product of GUI testing can foster the understanding of GUI tests.**

***Index Terms*—by-product, GUI test, video**

I. Introduction

Software and graphical user interface (GUI) have become an integral part of everyday life. As a result, humans use software throughout the day and have correspondingly high expectations of its design, usability, and user experience. For this reason, software and especially its GUI must be well and continuously tested to ensure that (GUI) defects are not delivered to customers. However, developers frequently experience and report difficulties with identifying and understanding defects and their sources due to incomplete and imprecise test reports [1], [2]. One crucial issue of test reports is their lack of dynamic details about interaction process and their individual steps to reproduce a GUI defect and identify its source [3]. In particular, the textual and thus static descriptions of interaction processes and their steps are often unclear and have to be modified most frequently during development [2]. In addition, missing links between test reports, the software (source code), and specification documents impede the work of developers as the pure test report lacks the important key events of the respective interaction process for reproduction [1]. The issue of missing or unclear information for reproducing a defect was identified as a core challenge of test reports by Zimmermann et al. [4] as well as Soltani et al. [5]. However, a test report "must make simple and unambiguous reproduction of a failure possible" [6, p. 329]. In most cases, several people with different technical and educational backgrounds have to deal with this test report, such as managers, developers, users and other stakeholders [6]–[8], which leads to the problems of inaccuracy, incompleteness, and thus limited reproducibility mentioned above. Hence, a test report is created and used by people with different interests and educational backgrounds.

We propose the provision of videos as supplementary material to static test reports addressing the issue of inaccuracy and incompleteness. Based on the video as a by-product approach by Karras et al. [9], [10], we propose an approach that integrates video generation with GUI testing to enrich static test reports (e.g., textual documentation, screenshots) with dynamic videos of interaction steps and key events. This type of video is a suitable documentation option for communicating the dynamic details of interaction processes on GUIs to a variety of people with different technical and educational backgrounds [9], [10]. In this work, we focus on helping *testers* produce a video as supplementary documentation of static GUI test reports and helping *developers* understand the GUI test through watching the generated video. Although we limit the target audience of this type of video to developers for now, we assume that these videos are of value even to other stakeholders due to their simplicity and ease of use [11]. This paper has the following contributions:

1) An approach for videos as a by-product of GUI testing that integrates video generation into GUI testing so that the test reports associated with test logs and screenshots are enriched with the generated videos.
2) An experiment showing that the generated videos and screenshots lead participants to the same effectiveness and efficiency in determining if there is a defect.
3) The insight that developers' preferences and opinions on using videos and screenshots in understanding GUI tests are different. Each kind of documentation has its strengths and weaknesses. Thus, videos and screenshots are both valid documentation options.

The rest of this paper is structured as follows: Section II lists

related work. Then the approach of videos as a by-product of GUI testing is explained in Section III. Section IV presents the experiment and the results. The last section concludes and proposes future work.

II. Related Work

Video has been proposed as a new medium for communication in Requirements Engineering (RE) over one decade. Brill et al. [12] have proposed a low-effort video production approach in requirements engineering. These videos help to validate requirements from the perspective of customers and requirements engineers. Schneider et al. [13] further developed this approach into the affordable video approach, in which the video is produced in affordable efforts to assist elicitation and validation in RE.

Different works combined videos as a documentation option with additional information, such as source code [14], [15] and interactions [9], [16]. Pham et al. [14], [15] recorded videos while GUI test executions to supplement test reports with the videos. These videos are specially coded for saving storage resource while storing additional information. In particular, the video is linked to the source code lines: The corresponding source code line is highlighted while video is playing.

Shi and Schneider [16] have enhanced the tool suite ScreenTracer from Pham et al. [14], [15]. Shi and Schneider have fixed the issue of capture delay [15] and proposed two concepts for highlighting GUI interactions: Code injection and frame manipulation. Tandun [17] has implemented the code injection concept in ScreenTracer. The GUI interactions are automatically highlighted with a red rectangle. In the updated ScreenTracer, the replay speed can be slowed down and the console output is linked to timestamps in the video. Tandun has conducted a semi-structured interview to ask ten testers in a company about opinions of the updated ScreenTracer. According to the coding results [18], the testers can use the tool suite to find the accurate position of a defect and use minimum time to understand a defect.

Karras et al. [9] have proposed the video as a by-product approach and applied it to prototyping. Based on their approach, videos are generated that demonstrate interaction sequences on hand-drawn and digitally created mockups as additional support for textual scenarios. These videos are produced as a by-product of digital prototyping by capturing and replaying interaction events of responsive controls without any implementation. In the experiment of Karras et al. [9], they found that such videos allow a slightly faster understanding of textual scenarios by developers compared to static mockups.

Using replay methods or videos to support the reproduction of a defect is in practice. Selenium [19] can be used for capturing the dynamic interactions and reproduce them. Test scripts can be created and then replayed in Selenium. Another tool Ranorex Studio records a video during the test and uses the video as part of the test report [20]. In the academia, Nass et al. [21] developed a tool called Scout which captures the test executions and replays them with instructions. Testers can add hints on Scout to make the tests more informative. Scout replays hints with highlighting over GUI elements, e.g., a rectangle that highlights the next GUI element in interaction.

Our work has the following innovations compared to presented related work: 1. We generate the video during GUI testing. This video is not for elicitation and validation, but for verification in RE. The video can be used as a communication medium between developers and testers. By watching the video, it can be checked if the understanding of testers about a requirement is the same as the understanding of developers. 2. The generated video from our approach is linked with test annotations and outputs, not just a video itself (such as in Ranorex's solution). This linking of artifacts helps developers and testers better understand GUI tests, as artifacts with strong interrelationships must be processed and related together to be better understood, as shown by Karras et al. [22], [23]. 3. Our approach generates video automatically during the test. This approach differs from the one from Nass et al. [21], [24], where testers need to add the additional information manually. In our approach, testers can generate videos by just running scripts without any other activities.

III. Approach

Based on previous work [15], [16], we systematically followed the video as a by-product approach by Karras et al. [9], [10] to develop and implement our approach and its concepts for videos as a by-product of GUI testing. Our approach produces videos during test execution and uses these videos for communication about requirements. This approach can be integrated into processes which contain development and execution of GUI tests.

Figure 1 uses FLOW notation by Stapel et al. [25] to explain how we apply the video as a by-product approach for GUI testing (in an agile development context). After receiving written requirements, developers start implementation and deliver working software (A). In agile development, testers develop tests (B) before implementation of functionalities. When testers have access to executable files with source code, they can develop more concrete test cases (B) and test the GUI of the software. Next, testers execute tests (C) and generate videos during the test run. A static test report documentation can be generated automatically. Then testers organize the test report (D) from the static documentation and the videos. Lastly, testers and developers communicate with each other by viewing the test report.

A. Video as a By-Product Approach

Our approach is designed to help developers understand defects better than using only static documentation. The **main goal** of our approach is: Developers can use videos to understand the reported GUI defects of a software. Concretely, developers can **1**. Familiarize themselves with contextual information of defects; and **2**. Clarify if the reported defect is an true defect.

We follow eight of the nine principles of the video as a by-product approach [9], [10] to revise GUI testing and to apply videos as a by-product. These eight principles are: P1 *Focus*,

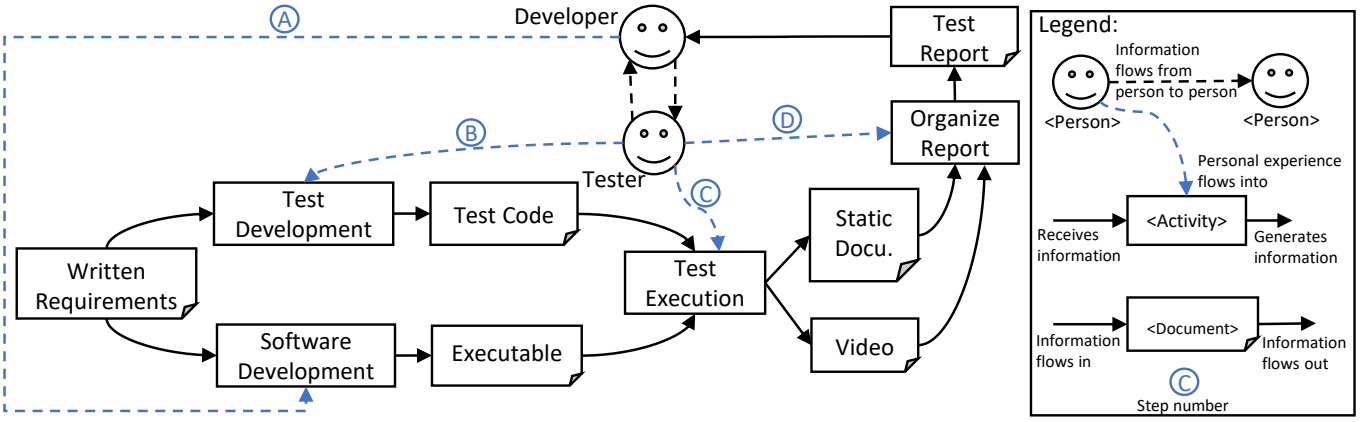

Fig. 1. FLOW diagram of our videos as a by-product of GUI testing approach.

P2 *Concurrency*, P3 *Parties involved*, P4 *Relief*, P5 *Separation*, P7 *Combination*, P8 *Fallback option*, P9 *Further use*. We do not apply the principle P6 *Technology* by Karras [10] as we do not focus on the use of mobile devices to generate videos but rather on personal computers.

Based on the eight principles and our goal, we developed three concepts for our approach.

- **Concept 1**: Video is generated during the test execution to describe a system appearance;
- **Concept 2**: Video is replayed with additional information (highlighted GUI interactions, step descriptions, command line output, etc.) to make the video easy to understand;
- **Concept 3**: Clickable texts of step descriptions and command line output are used in replay to make efficient defect analysis possible.

We explain how the concepts correspond to principles. Firstly, all three concepts are proposed with the focus on one practice, namely GUI testing (**P1**). A video is generated during the test execution (Concept 1). The video is produced (Concept 1) and used (Concepts 2 and 3) in agile development context (**P2**). Concept 1 helps testers to generate a video with existing test scripts in minimum steps (**P4**). Concepts 2 and 3 enable testers and developers (**P3**) to watch the video with linked additional information, which help them understand the test (**P4**). All concepts are profitable for testers and developers and no other roles are burdened (**P3** and **P4**), as Fig. 1 shows. Additional information is combined with the video (**P7**). Next, we want to separate the logic of producing and replaying video into two software components (**P5**). Video serves as supplementary material for static test reports. A traditional textual test report is by all means created as a fallback option (**P8**). Video as a by-product in GUI testing can be applied not only in internal communication between developers and testers, but also can be used in communication with customers (**P9**). However, the video must be adapted to solicit customers' feedback, which is part of our future work.

B. Technical Implementation

We implemented our concepts in a tool called ScreenTracer that generates a video during test execution. The video is replayed in a video player with clickable texts. Below, we describe how the concepts are realized.

1) Video Generation: The video is generated by running existing GUI test scripts (Concept 1), which are developed during the software development. We have used the tool ScreenTracer [15], [16] with existing NUnit[1] test projects to capture the videos. These test projects use Selenium WebDriver[2] to conduct GUI tests on a website. A test project contains a test suite, and the test suite contains many test cases. The test run executes these test cases, which are called in the main function.

To highlight the GUI interactions (Concept 2), a dynamic method changes behavior of the program: Every time the element is clicked or be typed in, the dynamic method draws a red rectangle before the interaction and hides the rectangle after the interaction. In the test script, testers write annotations (e.g. Console.WriteLine("Step 1: Log in");) to add additional textual information to the video. This textual information with the corresponding times is stored with the video. Additionally, we have added a function to select the capture area to capture only the test object and to save storing cost. All technical details of the ScreenTracer can be found in the works of Pham et al. [14], [15] and Tandun [17]. A tester needs to conduct three steps: 1. Select the test project; 2. Select the area to capture; 3. Click the capture button. After a test is completed, a video is generated.

2) Video Replay: Figure 2 shows how the video is replayed in a video player. The video player loads the video and the clickable text under the video. User can go to referred timestamp by clicking the blue text (Concept 3).

3) Limitation: As the driver of screen capturing is outdated in ScreenTracer, the generated video is choppy. For the sake

[1]https://docs.microsoft.com/en-gb/dotnet/core/testing/unit-testing-with-nunit

[2]https://www.nuget.org/packages/Selenium.WebDriver

of evaluation, we use the OBS studio to capture and store the video as an mp4 data during the same test run as ScreenTracer uses. The additional textual information with timestamps are also temporary manually created: A list of strings and the corresponding times are stored as a Web Video Text Tracks (vtt) file. The video player as a web page loads the mp4 data and the vtt file for replay. The produced video and the video player are in accordance with the concepts. We have published example videos, vtt files, and HTML source code of the video player in the dataset [26] for transparency.

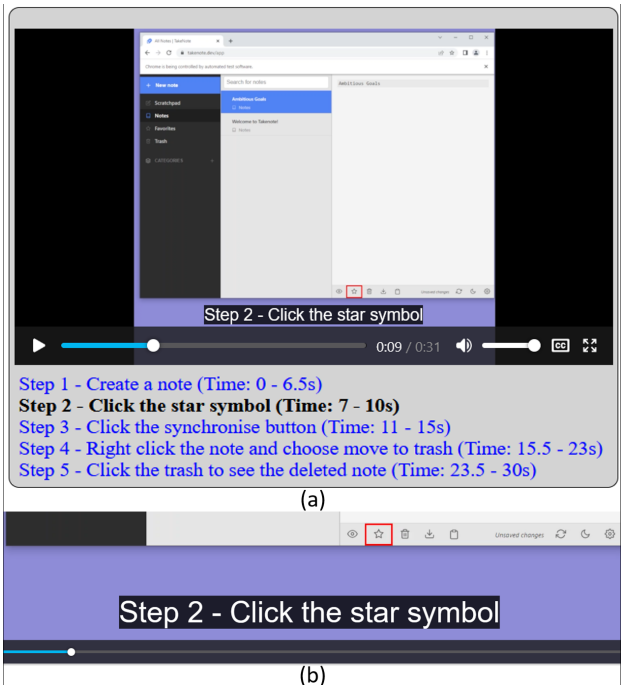

Fig. 2. HTML video player (a) and magnified GUI with highlight (b).

IV. Evaluation

We wanted to evaluate the effect of GUI test videos. According to video as a by-product approach for GUI testing, a defect is reported by testers and handed in over developers. We wanted to know if a GUI test video helps developers understand a defect. The reasons of the defect can be 1. developers did something wrong in the source code, i.e., the defect is true or 2. testers misunderstand the requirements and write the wrong test cases, i.e. the test case is incorrectly specified, the reported defect is false.

In our study, we compare videos with screenshots, as both visualize the GUI tests. Based on the goal template by Basili and Rombach [27], we concrete our evaluation goal as follows: **We analyze** GUI test videos **for the purpose of** distinguishing between true and false defects **with respect to** their effectiveness and efficiency **from the viewpoint of** the developers **in the context of** watching videos and screenshots. Here, effectiveness means that developers can find the right reason of the reported defect. Efficiency means that developers can find a reason in a short time. The participation of this study is voluntarily and without compensation.

We test the following two hypotheses: There is a difference in

H_1: efficiency
H_2: effectiveness

between the test video and the screenshots. The corresponding null hypotheses assume that there is no difference between the test video and the screenshots.

We ask the following research question (RQ):

> **RQ**: What are the strengths and weaknesses of test videos and screenshots in understanding GUI tests in the perspective of developers?

A. Study Design

In this experiment, we have one factor with two treatments (i.e. screenshots and a video). We use the paired comparison design [28], i.e., each participant watches not only screenshots but also videos. We choose to include four test cases, because 1. participants can conduct the experiment in an acceptable and affordable time, and 2. collect as much quantitative data as possible for statistical tests, and 3. check the preference of documentation for different test cases. The test case contains many test steps. Each step is documented as a textual description, which are linked with the video, as Fig. 2a shows. For each step, one screenshot is captured as comparison. The textual description is also set as the name to the screenshot. That means, for each case, there is one video and multiple screenshots. All videos and screenshots are available in our dataset [26].

We choose a web-based note app called TakeNote[3] as the test object for evaluation. This app is about note taking and maintaining, with which developers should be familiar. We also prepared a tutorial video of this app.

Figure 3 shows the general study design. After a short introduction of the experiment, participants are required to watch the source code folder structure and the tutorial video of the app. Then, participants are required to read the static documentation which contains the given requirement, additional information (if needed) for that requirement, related code lines, and the expected and the real results of the test case run. The textual documentation of test case 3 is shown as follows:

> **Given requirement**: If the user deletes a category permanently, the TakeNote App shall 1. permanently remove the category and 2. keep the notes which belonged to that category under folder Notes.
> The **related code** are lines 69 - 73 in:
> src/client/containers/ContextMenuOptions.tsx
> **Expected result**: The note "The solar system - Earth" and the note "The solar system - Jupiter" should not appear in the note list.

[3]https://github.com/taniarascia/takenote, demo available on https://takenote.dev/ (State 7-Jun-2023)

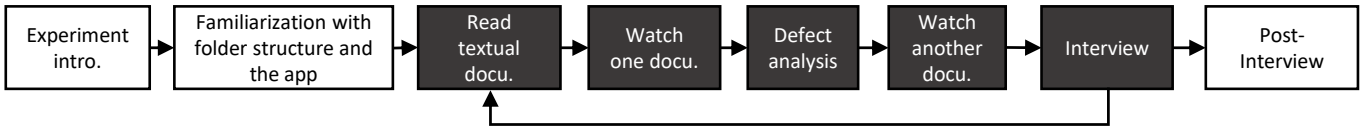

Fig. 3. General experiment process. Activities in black boxes are repeated five times. The first time is for learning the task. The further four times are for carrying out real tasks.

Real result: The note "The solar system - Earth" and the note "The solar system - Jupiter" appear in the note list.

Next, participants watch both types (i.e. the video and the screenshots) subsequently for each test case. They are required to watch one documentation type to select a reason of the reported defect in the phase *defect analysis* in Fig. 3 between three choices (true defect/false defect/not sure). During *defect analysis*, participants may read the source code. We measure the time taken (as **metric 1**) of the *watch one docu.* and the *defect analysis* (Fig. 3) to check the efficiency (H_1) of the watched documentation. We also count the number of correct reasons (as **metric 2**) to check the effectiveness (H_2) of the watched documentation. After the task of *defect analysis*, participants watch another documentation type. They are asked to list the strengths and weaknesses of each documentation (RQ), choose the preferred documentation and give reasons for the choice in the phase *interview*. At last, the experiment conductor tells participants the right answers of the reasons and the measured time. Participants are required to answer questions about effectiveness and efficiency (*post-interview*). We provide the questions in our dataset [26].

We divide participants into two groups and have a crossover design for the settings of the groups. For the phase *watch one docu.*, group A watches videos for test cases 1 and 2 and screenshots for test cases 3 and 4; group B watches screenshots for test cases 1 and 2 and videos for test cases 3 and 4.

The type of defects (true/false) is mixed in the study.

1) Test case 1 is about a true defect. A rendered horizontal line should be clearly visible in the dark mode but was not visible if the mode is changed from normal to dark mode.
2) Test case 2 is about a false defect. Testers misunderstand that the Takenote App should reset all personalized settings in sorting. They think the App should sort the notes in alphabetical order after logout and login again. According to the requirement, the settings are permanently stored and will not be reset. In the test run, the setting of "Sort by" was changed from *Last updated* to *Date Created*, and then logging out and in was conducted. Testers have reported a defect and they claim the notes should be sorted in alphabetical order as expected output.
3) Test case 3 is about a false defect. Testers misunderstand the deletion function of a category. They think that after deletion of a category permanently, the notes under that category should also be deleted. According to the requirement, the notes under that category should be kept under the main folder.
4) Test case 4 is about a true defect. As soon as the user changes the watching mode (editing or prewatch), the TakeNote App shall provide the user with the ability to visually determine which watching mode to switch to. The switch between editing and prewatch modes can be conducted by either clicking the symbols (shown as eye or notebook) or pressing the shortcut Ctrl+Alt+P. The Takenote App should change the eye symbol to the notebook symbol if Ctrl+Alt+P is pressed, but keeps unchanged.

B. Demographics

Eleven participants took part in the experiment, 6 for group A and 5 for group B. Two participants are female. All participants come from the same age group from 24 to 32 years old. Median age is 29. All have rich general development experience with the average of 10 years. For web frontend (HTML, JavaScript/TypeScript) specially: 4 are beginners, 5 have intermediate skill, 2 are advanced developers. Four participants (with identifiers P03, P05, P06, and P07, see also Tab. I) are working students, the other seven are industrial practitioners.

C. Result and Analysis

Table I shows the time taken (metric 1) and mark incorrect selections of defect analysis in gray cells. To test H_1 and H_2, we conducted the Mann-Whitney U test for these metrics, as Tab. II shows. To calculate the Mann-Whitney U test for metric 2, we follow the instructions from Bortz and Schuster [29] for small samples of shared rankings. For the sake of transparency, the individual calculation steps for this test are provided in the data set [26]. We set the significance level α to 0.05. As the test is conducted for two times for each metric respectively, we use the Bonferroni corrected $\alpha = 0.05/2 = 0.025$.

By calculating Cohen's d as effect size, we find a visible small difference in metric 1 and a visible medium difference in metric 2. However, the p values show that these differences have no statistical significance, probably due to a small sample. Hence, we cannot reject the null hypotheses of H_1 and H_2.

To answer the research question, we have firstly marked the relevant words in the answers of strengths and weaknesses. After that, we have pattern coded the marked words: 1. Summarize seven categories as different topics; 2. Code words as advantage or disadvantage for a category; 3. Count occurrence

TABLE I
QUANTITATIVE RESULTS (TIME IN SECONDS)

	Case 1	Case 2	Case 3	Case 4
Group A	Video	Video	Screenshots	Screenshots
P01	49	280	116	95
P03	87	323	265	133
P04	-	-	128	112
P06	55	378	133	328
P07	58	226	45	141
P10	91	152	56	16
Group B	Screenshots	Screenshots	Video	Video
P02	113	108	100	133
P05	46	148	20	25
P08	45	300	123	122
P09	193	321	299	370
P11	30	71	242	83

Gray background means that the corresponding participant did not select the right reason of the corresponding test case.
P04 has experienced technical issues for cases 1 and 2, so the results are omitted.

TABLE II
RESULTS OF MANN-WHITNEY U TEST

Metric	For Test Cases	U	p	Cohen's d
M1: Time taken for watching and analysis	1 & 2	8	.4009	0.5
	3 & 4	13	.7872	0.2
M2: # correct selections	1 & 2	7.5	.2207	0.8
	3 & 4	10	.3173	0.5

of advantages and disadvantages for each documentation and each category. Table III shows the coding result. We **answer RQ** with Tab. III and the following descriptions. The category *Visualization* is mentioned more than 40 times. Participants mention more advantages than disadvantages for videos, while more disadvantages than advantages for screenshots. Regarding category *Time*, participants have the opposite tendency. Long watching time is mentioned 10 times for video, while short watching time 9 times for screenshots. We have also observed opposite tendency in category *Completeness*. Videos seem to show participants details from the GUI test, but screenshots seem to lack details. In category *Identification & Navigation*, participants mention that the clickable step descriptions help navigation and the difficulty of locating relevant positions by using videos. Participants meant that the screenshots can help them to identify the problem more quickly but make navigation more difficult by using screenshots. The complete coding is available in our dataset [26].

D. Threats to validity

In the following, we present threats to validity, categorized according to Wohlin et al. [28].

In the study, participants watched videos and screenshots and then compared them. The order of watching may have an effect on perception and, therefore, on their feedback. To minimize this threat to the **construct** validity, participants watch one type for cases 1 and 2 and another type for cases 3 and 4 before the phase *defect analysis* (Fig. 3).

Participants would take time to learn how to use videos or screenshots to find the reason of the reported defect. A threat to the **construct** and **conclusion** validity is that the time measurement would be not accurate. To mitigate that, the experiment conductor taught participants how to use the video player to watch videos and how to navigate through screenshots in the phase of learning the task.

Another threat to time measurement is the familiarity of the project. If participants already knew the TakeNote project before the study, they could use less time analyzing the bug: this threatens the **conclusion** validity. Participants must not know this project before the study. We checked this when we asked for their consent.

The quantitative results show visible differences in metric 1 and metric 2, but without the statistical significance. The number of the participants and of the test cases in this study could be negligible to achieve the statistical significance. The generalization of the test results may be a threat to the **conclusion** validity.

Test cases 1 and 4 are about true defects of the TakeNote App. A participant could easily recognize the true defect without watching the documentation (screenshots/video), if the given requirement and the test case description (expected result and real result) were very similar. To mitigate this threat to the **construct** validity, we have used the *Detailed FunctionalMASTER* template from Rupp et al. [30] to formulate a requirement. The participants need to watch the screenshots or the video to understand a given requirement. Only when they understand the requirements, they can get familarized with the reported defect and find the reason.

In choosing the reason of reported defects, we provide an option "After looking the test-video/screenshots and the source code carefully, I am still not sure." We do not force participants to choose a reason (true or false defect). Participants seem not to guess a hypothesis because the experimenter encouraged them to select their subjective option. This mitigate the threat to the **construct** validity. Meanwhile, we can collect participants' real choices and use this collected data to check RQ2 (**conclusion** validity).

E. Discussion

1) Comparison between both documentations: A reviewer has pointed out that the screenshots seem to be a subset of the video, because the video consists of frames, some of which are exactly the screenshots. We argue that the screenshots and the video are different media and viewers interact with them differently. If the video was viewed by opening the frames as screenshots, the subset relation would be true. However, the screenshots are viewed in an image viewer, while the video is viewed in a video player on the HTML page (Fig. 2). A screenshot is captured automatically after the corresponding step is conducted: this shows the end of the corresponding step and the begin of the next step. Both documentations provide with step definition: as windows title which shows on the top of the image viewer (screenshots), as clickable subtitles which shows under the video player (video). Hence, both documentations present the test step in different ways.

TABLE III
ADVANTAGES AND DISADVANTAGES OF BOTH DOCUMENTATIONS

Category and meaning	Type	Frequency		Typical example from participants
Visualization: Mentioned documentation visualizes the test case run.	Video	pro	20	It shows the user interactions clearly
		contra	3	don't cover shortcuts interaction
	Screenshot	pro	9	clear screenshots of buttons
		contra	14	It's not that clear what the SW User is doing.
Time: The time which participants will use the mentioned documentation to watch or think.	Video	pro	1	it may give more time to think about the source of issue
		contra	10	it takes longer to watch
	Screenshot	pro	9	quick to watch
		contra	1	it takes time to understand what happens
Completeness: Mentioned documentation describe the test case run in a complete way.	Video	pro	6	showing all the details from the workflow
		contra	3	they might not be complete
	Screenshot	pro	4	Shows all the steps
		contra	9	Missing out on details
Identification & Navigation: Mentioned documentation helps participants identify the problem or nagivate to the problem position.	Video	pro	3	Clickable test steps helps navigation
		contra	3	must click pause to see wished position
	Screenshot	pro	9	Giving an idea where to look quickly
		contra	3	navigating takes more effort
Readability: The readability of mentioned documentation	Video	pro	0	-
		contra	1	dark mode making difficult to read
	Screenshot	pro	3	more readable of the details on images
		contra	0	-
Explanation: Mentioned documentation explains the test case run.	Video	pro	3	There is a text explanation at each video.
		contra	1	I don't know how the dark mode is activated.
	Screenshot	pro	1	Title of each screenshot helped me to understand
		contra	5	Missing out on ... explanation
Follow: Participants follow the mentioned documentation.	Video	pro	3	It's easier to follow the steps of the SW User.
		contra	4	difficult to follow without the audio.
	Screenshot	pro	0	-
		contra	1	Harder to follow the steps taken by the SW User.

Our study wanted to compare these two media and find any difference in efficiency and effectiveness.

2) About test case 2: In Tab. I, we notice that six participants made wrong selection (four of them were not sure) of the reason for defect analysis of test case 2. The reported defect of test case 2 is a false defect. The given requirement is "As soon as the user changes editor preferences, the TakeNote App shall be able to permanently store the updated preferences as local storage in the Web browser." In the test run, the setting of "Sort by" is changed from *Last updated* to *Date Created*. The misunderstanding from testers was two-fold: First, they thought the settings should be reset after logout and login again; second, they thought the notes should be sorted alphabetically by default.

However, six participants are confused about the "Sort by" setting of *Date Created*. P01 and P07 asked the experimenter if the date is a day or time. P02, P06, P10, and P11 were unsure if the "Sort by" setting of *Date Created* is descending or ascending. The experimenter did not answer them these questions before they made a decision, as participants can infer from watching the given documentation, reading the source code carefully, and thinking about that (as P08 did). The source code (src\client\utils\notesSortStrategies.ts, available in our dataset [26]) shows that the timestamp is used in the *Date Created* sorting. Both documentations show that the newest created note is ordered as the first one.

The complexity of this misunderstanding and the confusion of the sorting function may be the reasons of following facts of this case: 1. Four participants (P01, P02, P05, P03) were unsure if the reported bug is a true bug; 2. All participants (except P02, P09, and P11) use the most of time for test case 2 (See also Tab. I).

3) Audio in the test video: In our study, three participants expressed their need for audio in the test video. P04 wrote in test case 0 (the case for learning the task): "no audio (is wished: description of the usage, more understandable if given audio), someone may not concentrate on the video content without audio". P08 and P09 wrote similar opinions. This is worth further investigating, if added audio as explanations in the generated test video help developers understand a test better.

4) Suitability of video in communication: Our results show that videos and screenshots do not differentiate in effectiveness and efficiency. However, we argue that the videos are more suitable for communication in the development team. Cockburn [31] states that the video is more effective in "transferring ideas" than the paper. For complex cases like our test case 2, videos can show the test clearly (Visualization in Tab. III) and completely (Completeness in Tab. III). A video shows what really happened, while this information must be under some cases inferred from static documentation. Using videos reduces the effort of clarifying incorrect inference.

V. CONCLUSION AND FUTURE WORK

Developers cannot understand a static report of a GUI defect. One reason of that is the report is missing dynamic details. This work proposes to generate videos as a by-product of GUI testing as supplements to static test reports (e.g., texts and screenshots). The generated video is annotated with red

rectangles and linked with textual test descriptions. Testers produce the video by running the test. Developers watch the annotated video with clickable texts.

In our empirical study, we wanted to investigate if test videos and screenshots differ in terms of efficiency and effectiveness in understanding a GUI test. We also wanted to know the opinions about these two documentations from developers. According to the quantitative results, we cannot statistically prove the difference between two documentations in efficiency and cannot statistically prove the difference in effectiveness. Our pattern coding results show that the each documentation has its advantages and disadvantages. The frequency (Tab. III) indicated that both documentation supplement with each other, as a frequently mentioned advantage of a documentation can be a frequently mentioned disadvantage of another documentation. **The frequently mentioned advantages in categories *Visualization*, *Completeness* (Tab. III) confirm that the test video helps developers understand GUI tests.** Our videos as a by-product of GUI testing approach provides a better way to organize a test report. The report not only includes texts and screenshots, but also videos which link with these texts: Developers and testers can use this report to better understand the GUI test.

The study results also show that the preferences of these two documentations vary between the four test cases. We may assume that the preference is dependent on the type of the test case. We will analyze the collected preference reasons in the interview to investigate this assumed dependence. Furthermore, we will update the driver of screen capturing and integrate this driver into the ScreenTracer.

ACKNOWLEDGMENT

This work is funded by Deutsche Forschungsgemeinschaft (DFG) - Project number 289386339 (ViViUse). We thank the reviewers for their valuable comments.

REFERENCES

[1] J. Aranda and G. Venolia, "The secret life of bugs: Going past the errors and omissions in software repositories," in *31st International Conference on Software Engineering*. IEEE, 2009.

[2] D. Wang, Q. Wang, Y. Yang, Q. Li, H. Wang, and F. Yuan, "'Is It Really a Defect?' An Empirical Study on Measuring and Improving the Process of Software Defect Reporting," in *2011 International Symposium on Empirical Software Engineering and Measurement*. Banff: IEEE, Sep 2011.

[3] A. M. Memon, "GUI testing: pitfalls and process," *Computer*, vol. 35, no. 8, Aug. 2002.

[4] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What Makes a Good Bug Report?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, Sep. 2010.

[5] M. Soltani, F. Hermans, and T. Bäck, "The significance of bug report elements," *Empirical Software Engineering*, vol. 25, no. 6, Nov 2020.

[6] A. Spillner, T. Roßner, M. Winter, and T. Linz, *Praxiswissen Softwaretest: Testmanagement : Aus- und Weiterbildung zum Certified Tester - Advanced Level nach ISTQB-Standard*. Heidelberg: dpunkt, 2014.

[7] E. Dustin, J. Rashka, and J. Paul, *Automated software testing: introduction, management, and performance*. Reading, Mass.: Addison-Wesley, 1999.

[8] K. Moran, M. Linares-Vásquez, C. Bernal-Cárdenas, and D. Poshyvanyk, "Auto-completing bug reports for android applications," in *10th Joint Meeting on Foundations of Software Engineering*. ACM, Aug 2015.

[9] O. Karras, C. Unger-Windeler, L. Glauer, and K. Schneider, "Video as a by-product of digital prototyping: Capturing the dynamic aspect of interaction," in *25th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2017.

[10] O. Karras, *Supporting Requirements Communication for Shared Understanding by Applying Vision Videos in Requirements Engineering*. Hannover: Logos Verlag Berlin GmbH, 2021.

[11] ——, "Software professionals' attitudes towards video as a medium in requirements engineering," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2018.

[12] O. Brill, K. Schneider, and E. Knauss, "Videos vs. use cases: Can videos capture more requirements under time pressure?" in *Requirements Engineering: Foundation for Software Quality*, vol. 6182. Springer, 2010.

[13] K. Schneider, M. Busch, O. Karras, M. Schrapel, and M. Rohs, "Refining vision videos," in *Requirements Engineering: Foundation for Software Quality*, vol. 11412. Essen: Springer, 2019.

[14] R. Pham, H. Holzmann, K. Schneider, and C. Bruggemann, "Beyond plain video recording of GUI tests: Linking test case instructions with visual response documentation," in *7th International Workshop on Automation of Software Test (AST)*. Zurich: IEEE, Jun. 2012.

[15] R. Pham, H. Holzmann, K. Schneider, and C. Brüggemann, "Tailoring video recording to support efficient GUI testing and debugging," *Software Quality Journal*, vol. 22, no. 2, Jun. 2014.

[16] J. Shi and K. Schneider, "Creation of human-friendly videos for debugging automated gui-tests," in *Testing Software and Systems*. Springer, 2021, p. 141–147.

[17] M. A. Tandun, "Report-video production for quick bug-finding in the web applications," Hannover, Jan 2022, [Bachelor Thesis]. [Online]. Available: https://www.pi.uni-hannover.de/fileadmin/pi/se/Stud-Arbeiten/2022/BA_Tandun_2022.pdf

[18] J. Shi, K. Schneider, M. Tandun, and O. Karras, "Supplementary material for evaluating screentracer among testers," Jan 2023. [Online]. Available: https://zenodo.org/record/7522978

[19] Software Freedom Conservancy, "About Selenium," https://www.selenium.dev/about/, 2022, [Accessed Jun-2023].

[20] B. Chauvin, "Video reporting in ranorex studio," https://www.ranorex.com/blog/video-reporting/, 2020, [Accessed Jun-2023].

[21] M. Nass, E. Alegroth, and R. Feldt, "Augmented testing: Industry feedback to shape a new testing technology," in *International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. Xi'an: IEEE, Apr 2019.

[22] O. Karras, A. Risch, and K. Schneider, "Interrelating Use Cases and Associated Requirements by Links: An Eye Tracking Study on the Impact of Different Linking Variants on the Reading Behavior," in *22nd International Conference on Evaluation and Assessment in Software Engineering*, 2018.

[23] O. Karras, A. Risch, and J. Klünder, "Linking Use Cases and Associated Requirements: A Replicated Eye Tracking Study on the Impact of Linking Variants on Reading Behavior," *Journal of Software Engineering Research and Development*, vol. 9, pp. 5–1, 2021.

[24] M. Nass, E. Alégroth, and R. Feldt, "On the Industrial Applicability of Augmented Testing: An Empirical Study," in *International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. Porto: IEEE, 2020.

[25] K. Stapel, K. Schneider, D. Lübke, and T. Flohr, "Improving an industrial reference process by information flow analysis: A case study," in *Product-Focused Software Process Improvement*. Springer, 2007.

[26] J. Shi, O. Karras, M. Obaidi, and M. Tandun, "Supporting Data Set for Paper 'Can Videos as a By-Product of GUI Testing Help Developers Understand GUI Tests?'," 2023. [Online]. Available: https://doi.org/10.5281/zenodo.8162396

[27] V. Basili and H. Rombach, "The tame project: towards improvement-oriented software environments," *IEEE Transactions on Software Engineering*, vol. 14, no. 6, 1988.

[28] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer, 2012.

[29] J. Bortz and C. Schuster, *Statistik für Human- und Sozialwissenschaftler*, ser. Springer-Lehrbuch. Springer, 2010. [Online]. Available: https://doi.org/10.1007/978-3-642-12770-0

[30] C. Rupp and die SOPHISTen, *Requirements-Engineering und -Management: das Handbuch für Anforderungen in jeder Situation*, 7th ed., ser. Hanser eLibrary. München: Carl Hanser Verlag, 2021.

[31] A. Cockburn, *Agile Software Development*. Addison Wesley, 2001.