Received: Added at production | Revised: Added at production | Accepted: Added at production
DOI: xxx/xxxx

ARTICLE TYPE

Organizing GUI Tests from Behavior-Driven Development as Videos to Obtain Stakeholders' Feedback

Jianwei Shi[1] | Jonas Mönnich[2] | Jil Klünder[1] | Kurt Schneider[1]

[1]Leibniz University Hannover, Software Engineering Group, Hannover, Germany
[2]imbus AG, Brunswick, Germany

Correspondence
Jianwei Shi, Leibniz University Hannover, Software Engineering Group, Hannover, Germany.
Email: jianwei.shi@inf.uni-hannover.de

Present address
Welfengarten 1, 30167 Hannover, Germany

Abstract

Demonstrating software early and responding to feedback is crucial in agile development. However, it is difficult for stakeholders who are not on-site customers but end users, marketing people, or designers, etc. to give feedback in an agile development environment. Successful Graphical User Interface (GUI) test executions can be documented and then demonstrated for feedback. In our new concept, GUI tests from Behavior-Driven Development (BDD) are recorded, augmented, and demonstrated as videos. A GUI test is divided into several GUI unit tests, which are specified in Gherkin, a semi-structured natural language. For each GUI unit test, a video is generated during test execution. Test steps specified in Gherkin are traced and highlighted in the video. Stakeholders review these generated videos and provide feedback, e.g., on misunderstandings of requirements or on inconsistencies. To evaluate the impact of videos in identifying inconsistencies, we asked 22 participants to identify inconsistencies between (1) given requirements in regular sentences and (2) demonstrated behaviors from videos with Gherkin specifications or from Gherkin specifications alone. Our results show that participants tend to identify more inconsistencies from demonstrated behaviors which are not in accordance with given requirements. They tend to recognize inconsistencies more easily through videos than through Gherkin specifications alone. The types of inconsistency are three-fold: the mentioned feature can be incorrectly implemented, not implemented, or an unspecified new feature. We use a fictitious example showing how this feedback helps a product owner and her team manage requirements. We conclude that GUI test videos can help stakeholders give feedback more effectively. By obtaining early feedback, inconsistencies can be resolved, thus contributing to higher stakeholder satisfaction.

KEYWORDS
GUI test, video, BDD, feedback

1 | INTRODUCTION

Software requirements are shaped through communication between stakeholders. Stakeholders[1] are people or organizations who have an influence on the requirements or are impacted by the system. They can be customers, developers, end-users, user experience (UX) designers, or marketing people who express opinions about a software system. A common issue is the insufficient communication among stakeholders in large-scale software projects[2,3] and in global software development[4,5]. This can lead to misunderstandings regarding the requirements. Thus, the implemented software can have wrong or missing functionality. This, in turn, leads to dissatisfied stakeholders or customers[2].

A possible solution is that developers demonstrate the software under development and get feedback from stakeholders. This feedback can be beneficial, because it comes from other perspectives and could also play a significant role on clarifying requirements. Potential end-users can give feedback on usage problems and suggest new features or extensions[6]. UX designers can give some feedback on quality related requirements, e.g., usability of software[7]. Moreover, marketing people can see the demonstrated software's GUI to identify inconsistencies with the corporate design.

In agile software development, continuous feedback is realized by regular demonstrations of the software in review meetings, e.g., in sprint review meetings[8]. This feedback can be addressed in future iterations[9,10]. However, only the on-site customer and the development team attend such meetings, and other stakeholders cannot give feedback this way. As argued above, other stakeholders can also give valuable feedback and should thus be involved in the process. Unfortunately, having more people attending the review meetings is often impossible due to large and diverse groups of stakeholders, who have other responsibilities. In this paper, we want to address this problem by presenting an approach to obtain feedback from diverse stakeholders. We propose to record graphical user interface (GUI) tests on video, augment this video with further explanations (if needed), and present this video to stakeholders who can give feedback without actually using the software themselves.

The video can also help a product owner quickly gain an overview on the current status of the project. In agile software development, a product owner wants to see a result in the presentation in a review meeting (e.g. sprint review meeting). According to private communication with some product owners, they want to see a successful demonstration without occasional failures due to imperfect code. The proposed video can be shown directly without running the software under development, which saves time for product owners, developers, and testers in the review meeting.

The aim of this paper is to use existing successful GUI tests to create and use videos to obtain feedback from stakeholders. A video is recorded during a slowed down GUI test run while GUI interactions are highlighted in the video. The resulting video is slowed down sufficiently for human observers to follow. In the video, the GUI control element under interaction is highlighted to show the test step with inputs. We provide stakeholders with the video to obtain feedback. We apply this concept in Behavior-Driven Development (BDD), an agile software development method. In BDD, the tests are first specified in a natural language and then developed in automated tests[11]. We propose to create the videos using the automated tests, and to explain the videos using the corresponding specifications.

Throughout this paper, we use the following fictional example to illustrate how our approach applies in BDD processes and contributes to software evolution:

Jenny is a product owner who has rich IT-security experience. She needs to take care that the software is implemented as already documented according to the communication with customers. She regularly holds sprint review meetings, where she sees the demonstration videos of already implemented functionalities. She then gives feedback on the demonstration and plans for the requirements for the next sprint. She also frequently obtains feedback from stakeholders regarding the current status of the software under development. However, the stakeholders are busy with their daily business and do not always have time to communicate with her. She tries to obtain stakeholders' feedback by showing them the demonstration videos and asking for their comments.

1.1 | Context of This Article

This article is an extension of Shi et al.[12] from the International Conference on Software and System Processes (ICSSP). The extended contribution points are highlighted in bold in the following paragraph.

In this article, we make the following contributions:

1. We provide supplementary steps beside the regular agile processes to involve stakeholders in the development. **We update the process descriptions and mark the supplementary steps clearly in the regular agile processes.** (Section 3)
2. We divide a GUI test into several GUI unit tests and organize them in a connection graph. Stakeholders see the graph and choose a continuous GUI test flow to watch. (Sections 3 and 5)
3. We evaluate the effect of videos with regard to finding inconsistencies between requirements and the demonstrated software from the stakeholders' perspective. (Section 5)
4. **Throughout the paper, we incorporate the fictional example of Jenny. We state how she gives feedback in the review meeting by viewing the videos (Section 3.3). Based on the coded results of inconsistencies identified by our participants, we show the activities through which she obtains and summarizes the feedback from stakeholders (Section 5.6).**
5. **We discuss the applicability of our approach by using a test framework called "Robot Framework", which is currently financed by more than 50 organizations[†] and popular among industrial practitioners.** (Section 5.5)

[†] https://robotframework.org/foundation/ Accessed: 1-Nov-2023

2 | RELATED WORK

We list the works which reveal the necessity of involving stakeholders in testing. Tests are written based on the requirements specification. Requirements should be written in a way that makes the tests easily interpretable. Hence, the alignment of requirements with tests is considered in this section. Lastly, we list works that use videos as a medium to communicate requirements.

An industrial case study from Mäntylä et al.[13] shows that not only testers are involved in testing, but also people who have a close relationship with customers. Sales personnel tend to discover additional relevant defects from the customer perspective. Chawani et al.[14] investigate the participation of stakeholders in software development for health care in Malawi. By inviting nurses as end-users during testing, requirements regarding the data format and user interface design were elicited. The managers of the health center and the ministry of health saw a system demonstration and provided feedback in more detail. In this investigation, feedback from various stakeholders complemented each other to generate concrete requirements.

The necessary practice of aligning requirements with tests is discussed in many studies[15,16,17,18]. Graham[16] has listed seven myths about common misunderstandings about requirements and tests. She argues that the requirements should be formulated as clear as possible, so that they can serve as a basis for test specifications (cf. myths 3 and 6). Ramesh and Jarke[15] have interviewed software companies about the requirements. From the results of the interviews, requirements are mapped to the tests to ensure the testability of the requirements. Bjarnason et al.[17] propose that testers check the testability of requirements. Most of their interviewees find the review of requirements by testers to be meaningful, because the review activates communication about requirements and thus improves their quality. Furthermore, Bjarnason et al.[18] conducted an industrial case study to investigate the benefits and challenges of using test cases as requirements. They summarize that this practice can help people in different roles within a company to communicate about requirements and to align their goals. A challenge of involving customers is that customer competence about GUI and quality requirements is required for communicating the related requirements.

To gain a shared understanding of requirements, Stangl and Creighton[19] propose the usage of paper GUI sketches to create videos. During the video creation, a executable GUI is not yet implemented. All GUI sketches are in prototypes. In their tool, the use cases are connected with corresponding GUI elements (e.g. an input field). Their tool could automatically regenerate the video when the GUI design is changed. For evaluation, Stangl and Creighton planned to conduct an industrial case study. However, we do not find that study to the best of our knowledge. Pham et al.[20] propose videos of GUI test executions for debugging purposes. The videos are captured during test execution, while a matching relationship between video and test code is created. A side-by-side viewer is used to replay the GUI interactions and corresponding lines in the test code. In the context of a qualitative evaluation in one company, the two interviewed developers believed that videos can help with debugging. Shi and Schneider[21] follow the concept of Pham et al.[20] and suggest the highlighting of GUI interactions. In the video, the GUI element under interaction (e.g., button or entry field) is accentuated by a colored border. Tandun[22] implements this highlighting concept in a software suite that is used to capture and replay a GUI test run. The effect of the video in debugging has been investigated among ten testers in a company. A semi-structured interview has been conducted to collect subjective opinions of the shown videos. Shi et al.[23] have coded these opinions. According to the coding results, participants mentioned that the highlighting function can help to reveal the position of defects accurately and explicitly.

Using the implementation[22], Shi et al.[24] have proposed the "videos as a by-product of GUI testing" approach and conducted a user study among ten participants acting as developers. In the study, the developers have to analyze the reported defect either by videos or screenshots from a test case. Each developer has four test cases and analyzes the defect through videos in two cases and through screenshots in the other two cases. The study has two groups: the first group analyzes the defect by videos in the first two cases and by screenshots in the last two cases; the second group analyzes the defect by screenshots in the first two cases and by videos in the last two cases. The quantitative results do not prove the difference in efficiency or effectiveness between videos and screenshots statistically. In the interview, developers express complementary advantages and disadvantages of videos and screenshots.

The **novel contributions** in this paper are: 1. We use videos for alignment of requirements with testing; 2. Videos are viewed not only by developers[20,21,24], but also by stakeholders, who are end-users, designers, developers, or testers; 3. We design an experiment to compare a classic (textual specifications) with a supplemented solution (textual specifications, videos and a connection graph). To check the effect of the supplemented documentations, we ask participants to recognize inconsistencies from demonstrated behaviors which are not in accordance with given requirements. This kind of dedicated experiment is missing in a similar earlier study from Stangl and Creighton[19].

3 | METHODOLOGY

A GUI test is a concrete representation of how a software should be used and what behavior is expected. When GUI tests are demonstrated to stakeholders, they may give feedback or modify their requirements. Hence, we propose our basic concept: **While a GUI test is executing, the execution is captured as a demonstration video. Meanwhile, the begin and end timestamps of each GUI interaction is linked with the demonstration**

video. Such a demonstration is not a test in itself but rather a system demonstration to obtain feedback. If an agile development team writes GUI tests and updates them regularly during development, these tests can be used for demonstration in order to solicit feedback and validate requirements. Behavior-Driven Development is an agile development method, in which the acceptance criteria are defined at first, and tests are maintained during the development. We want to apply this concept in BDD and ask our research question.

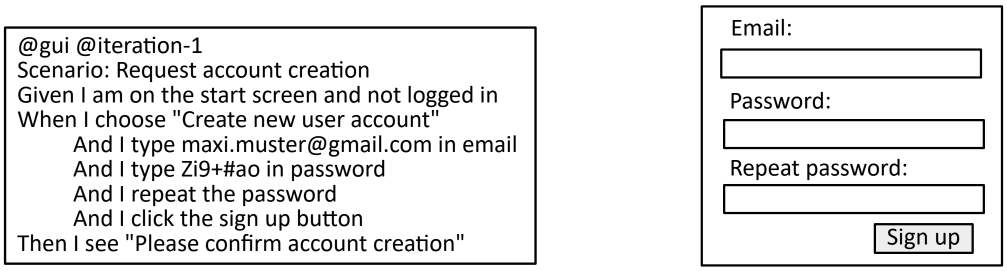
Research Question: How can existing GUI tests be used to create videos for obtaining feedback during Behavior-Driven Development?

3.1 | Gherkin specification and GUI Unit Tests in Behavior-Driven Development

We apply the proposed concept in Behavior-Driven Development (BDD), an agile development method.

Acceptance criteria contain scenarios which are specified in Gherkin. The Gherkin text of a scenario corresponds to the test case elements: *Given* (preconditions), *When* (actions with inputs), *Then* (expected results). An example: "*Given* I am on the page of 'duckduckgo.com' *When* I enter 'University' in the search field *And* I click the search button near the search field *Then* I see a list of links with short textual information". These Gherkin specifications are stored in feature files.

From the set of BDD tests, we use the automated GUI tests to create videos. In BDD, unit tests are implemented and automatically executed. We employ unit tests within the GUI to create videos. We call these tests "GUI unit tests" and propose the following definition:
A GUI unit test is a GUI test that interacts with logically related graphical control elements on the GUI. Figure 1 shows an example of a GUI unit test and the corresponding GUI.



```
@gui @iteration-1
Scenario: Request account creation
Given I am on the start screen and not logged in
When I choose "Create new user account"
        And I type maxi.muster@gmail.com in email
        And I type Zi9+#ao in password
        And I repeat the password
        And I click the sign up button
Then I see "Please confirm account creation"
```

FIGURE 1 Gherkin specification of a GUI unit test and the GUI after choosing "Create new user account".

In BDD, a feature file contains many scenarios, like the one in Fig. 1. One scenario specifies one GUI unit test. For each GUI unit test, a video is captured during test execution.

3.2 | General Application in BDD

Figure 2 depicts the supplemented BDD processes as a FLOW diagram. We use the FLOW notation by Stapel et al.[25]

We explain the activities based on the standard BDD steps according to Smart[11]. Our supplementations are marked as bold text in the following list. A sub-item means that the supplement is an extension of the standard BDD step of the corresponding item.

1. Business analyst, developers and testers come together to talk about requirements;
2. Business analyst, developers and testers define and write/update acceptance criteria;
3. **The business analyst creates a connection graph (e.g. Fig. 3) that shows relationships between scenarios (i.e., GUI unit tests);**
4. According to written acceptance criteria, developers develop the software, while testers write BDD unit tests. During this step:
   GUI unit tests are implemented according to Gherkin specifications. The specifications are stored in feature files;
5. The written tests are automated and run regularly. During this step:
   Executions of GUI unit tests are captured. The test execution should be modified automatically, i.e., the test execution is slowed down and interactions in the test are highlighted. The modifications follow the concept from Shi and Schneider[21].
   Test results are generated from a test automation framework and shown to the development team;

6. **The videos and the aforementioned graph are presented to stakeholders using our *GUI Unit Test Viewer* application (e.g. Fig. 4). A playlist of videos is organized, so that the stakeholders see a consolidated video. In the consolidated video shown in *GUI Unit Test Viewer*, the stakeholders can refer to corresponding specifications (scenarios). Stakeholders' feedback can be obtained during the presentation;**
7. **The obtained stakeholder feedback is considered in a review meeting, then the process starts again from step 2.**

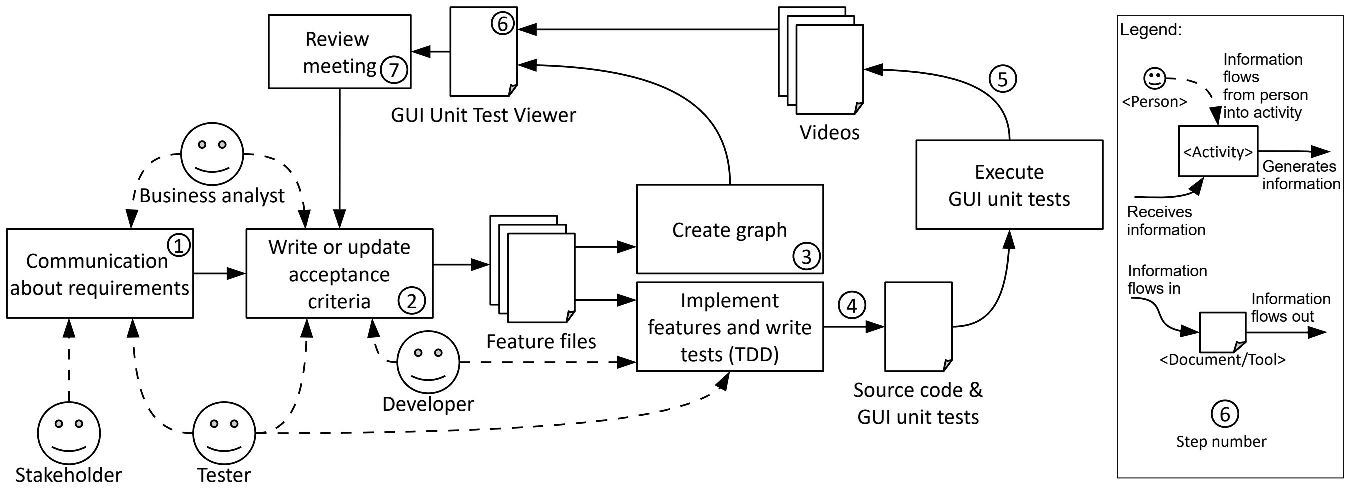


FIGURE 2 FLOW diagram of the supplemented BDD processes. Note that for activity "Review meeting" and "Create graph", all roles could participate: the dashed lines between all roles and these activities are omitted.

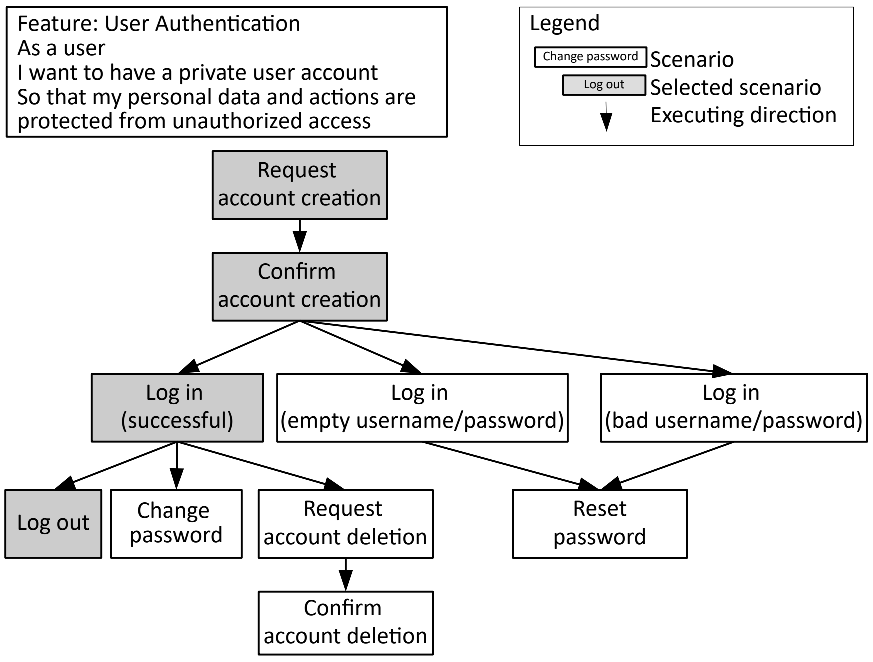
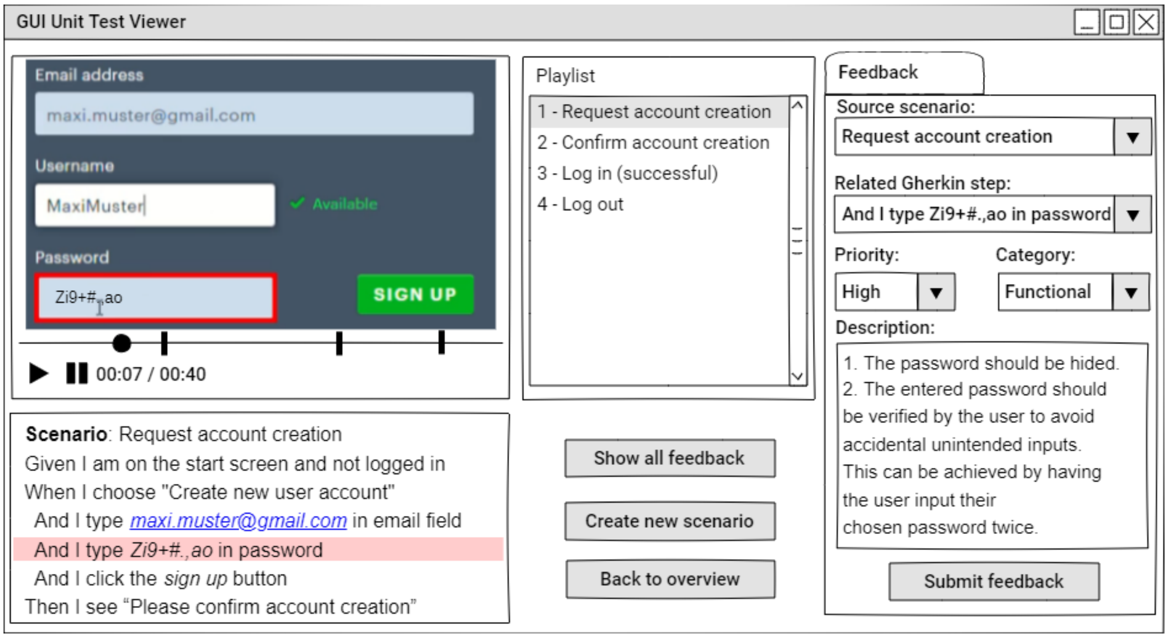

Feature: User Authentication
As a user
I want to have a private user account
So that my personal data and actions are
protected from unauthorized access
Legend
Change password Scenario
Log out Selected scenario
Executing direction
Request account creation
Confirm account creation
Log in (successful)
Log in (empty username/password)
Log in (bad username/password)
Log out
Change password
Request account deletion
Reset password
Confirm account deletion

FIGURE 3 Mockup of the connection graph.

3.3 Application in Jenny's team

Recently, Jenny's team received a task to develop a web application for film ratings. Jenny's team is using BDD in SCRUM. In the kick-off meeting, Jenny, representatives of users, developers and testers are present. They talk about the web application and formulate acceptance criteria in Gherkin. In the team internally, the team members communicate with each other about requirements in Gherkin. After two weeks, the first sprint is finished and Jenny holds a sprint review meeting. In this meeting, the team shows Jenny a connection graph (Fig. 3) of currently implemented scenarios for the feature "User Authentication".

Jenny is glad that so many scenarios are already implemented. She wants to see a user journey where a user has successfully registered, logged in and logged out. She selects these scenarios (the gray rectangles in Fig. 3) and is redirected to the video view (Fig. 4) of the GUI Unit Test Viewer.



1. The password should be hided.
2. The entered password should be verified by the user to avoid accidental unintended inputs. This can be achieved by having the user input their chosen password twice.

Scenario: Request account creation
Given I am on the start screen and not logged in
When I choose "Create new user account"
And I type maxi.muster@gmail.com in email field
And I type *Zi9+#.,ao* in password
And I click the *sign up* button
Then I see "Please confirm account creation"

FIGURE 4 Mockup of GUI Unit Test Viewer.

Jenny also sees that the password is shown explicitly in scenario "Request account creation" (See Fig. 3). This is not desired by stakeholders. Therefore, the next iteration should mask the password with asterisks (***). She also indicates that the password should be verified to prevent accidental unintended inputs according to quality requirement category *user error detection* in Software Product Quality (ISO 25010).

In addition, she also wants to see the edge cases where the system handles unsuccessful log in. She selects "Log in (empty username/password)" and "Log in (bad username/password)" and then sees the videos through GUI Unit Test Viewer. From her experience in IT-Security, she understands that the error message "bad username/password" not only indicates incorrect input but can also suggest vulnerabilities for a potential attack. She states that the unsuccessful logging in must consider the SQL injection risk. Therefore, a new scenario should be added: Log in (unsuccessful, SQL Injection).

After she gives feedback about IT-Security, she notices that the password should be strong enough to prevent attacks. Back to the scenario "Request account creation", she tells the team that they need to add a scenario called "Request account creation (weak password)" which takes a simple password as the Gherkin step: "And I type 123456 in password".

4 EVALUATION

In this evaluation we focus on the stakeholders' perspective. We are interested in whether videos help stakeholders provide feedback on the software. Based on the goal template by Basili and Rombach[26], we specify our evaluation goal as follows:

Goal definition:
We analyze videos of every single GUI unit test and a connection graph of these GUI unit tests
for the purpose of evaluation
with respect to the effectiveness of inconsistency identification between given requirements and videos
from the viewpoint of the potential stakeholders such as end-users, designers, developers, or testers
in the context of a controlled online experiment with Prolific participants in demonstrating account management functions of an online website.

We planned to involve participants via the online platform Prolific (https://www.prolific.co/researchers). We have chosen a film rating platform Letterboxd (https://letterboxd.com/) as the online website. The website should be easy understandable by the participants who have internet surfing experience.

4.1 | User Study Design

Based on the evaluation goal, we ask the following evaluation questions (EQ):

- EQ_1: How can videos and the connection graph of GUI unit tests support a precise identification of inconsistencies between requirements and the developed software?
- EQ_2: How can videos and the connection graph of GUI unit tests facilitate giving feedback for stakeholders?

To answer these evaluation questions, we give all participants the same textual requirements to unify their understanding. The behaviors of GUI unit tests are then demonstrated to the participants as Gherkin specifications or as Gherkin specifications with a video. The participants are asked to list inconsistencies between given requirements and the demonstrated behaviors. For example, if "delete account" is specified in the given requirements, but "deactivate account" is shown in the video, this difference is an inconsistency. Our second author first familiarized himself with an English website that he had selected. He then specified behaviors for the account management section of the website as GUI unit tests in Gherkin. Afterwards, he specified those same functionalities in regular sentences (as given requirements) while introducing several inconsistencies. Our first author then checked the given requirements and Gherkin specifications. Selecting these functionalities allowed us to (1) describe all functions as textual requirements in less than 350 words and (2) describe each function in Gherkin texts of no more than 12 lines and in video of no more than 38 seconds. Thus, long viewing time due to lengthy texts or videos has been avoided. We prepared ten scenarios (i.e., GUI unit tests) for the evaluation, e.g., creating an account, resetting a password, changing a profile picture. We want to investigate the effect of the videos and graph compared to the Gherkin specifications. Hence, participants are separated into two groups: 1. The control group receives only the Gherkin specifications, as those are available and up-to-date in BDD per default; 2. The experimental group receives the Gherkin specifications and, in addition, the videos and connection graph of the GUI unit tests, as we propose in Sec. 3.

Table 1 shows the designed metrics for EQ_1 and EQ_2. The metrics M_1, M_2 are calculated by counting inconsistencies from collected answers and are thus objective. The metrics M_3, M_4, M_5 measure subjective perceptions and are collected using an 8-point Likert scale, because (1) given an even number of options, participants should have a clear opinion acting as customers; and (2) eight options allow the participants to give their opinions precisely.

We are interested in testing the following hypotheses, which are shown in Tab. 1. The corresponding null hypotheses assume that there is no difference between both provisions in the respective metrics.

Figure 5 shows the steps of the experiment. In advance, participants should understand the Gherkin specifications. Therefore, Gherkin is explained in an example, followed by two multi-select multiple-choice questions to test comprehension. The questions are: "1. Which of the following keywords are always included in a scenario (according to the text above)? 2. Which of the following statements is/are correct?" Afterwards, the textual requirements (in regular sentences, not in Gherkin) are shown, followed by one multi-select multiple-choice question: "Which of the following statements is/are correct?" If participants are unable to answer the test questions correctly, the experiment must not be continued, as we assume that they have not understood the basics.

Next, the GUI unit tests are demonstrated. Participants in both groups receive the Gherkin specifications for every scenario. The experimental group receives one connection graph before seeing the first scenario. This connection graph conveys relationships between the ten scenarios by displaying some exemplary execution sequences (as shown in Fig. 3). After that, the GUI unit tests are demonstrated (1) through Gherkin specifications for the control group, (2) through Gherkin specifications and GUI unit test videos for the experimental group. At the end of each

TABLE 1 Metrics for the Evaluation Questions and corresponding Hypotheses

EQ	Metrics	**Hypotheses Begin:** There is a difference in ...
EQ_1	M_1: Number of correctly identified inconsistencies	the number of correctly identified inconsistencies (H_1)
	M_2: Number of incorrectly identified inconsistencies	the number of incorrectly identified inconsistencies (H_2)
EQ_2	M_3: Average certainty of mentioning inconsistencies (from 1: very uncertain to 8: very certain)	the average certainty of giving inconsistencies (H_3)
	M_4: Difficulty of identifying inconsistencies (from 1: very easy to 8: very difficult)	the difficulty of identifying inconsistencies (H_4)
	M_5: Understanding of development status (from 1: very limited to 8: very extensive)	the understanding of the development status (H_5)
		Hypotheses End: ... between the provision of Gherkin specifications and the provision of Gherkin specifications, connection graph, and videos.

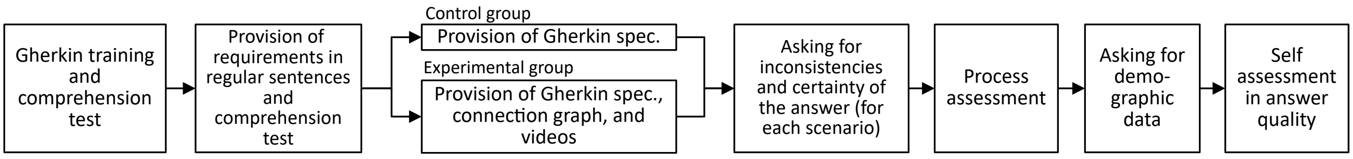
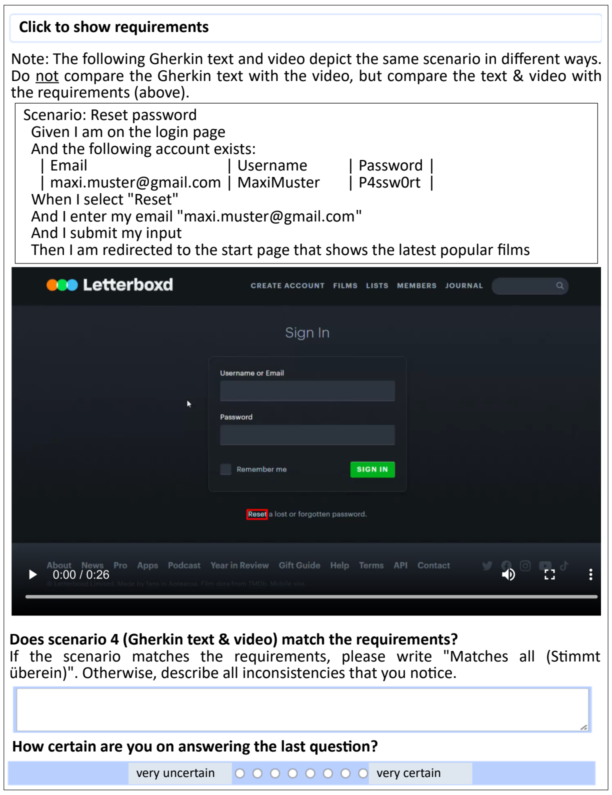

Gherkin training and comprehension test
Provision of requirements in regular sentences and comprehension test
Control group
Provision of Gherkin spec.
Experimental group
Provision of Gherkin spec., connection graph, and videos
Asking for inconsistencies and certainty of the answer (for each scenario)
Process assessment
Asking for demo-graphic data
Self assessment in answer quality

FIGURE 5 General experiment design. (Note: spec. means specifications)


Click to show requirements

Note: The following Gherkin text and video depict the same scenario in different ways. Do not compare the Gherkin text with the video, but compare the text & video with the requirements (above).

```
Scenario: Reset password
  Given I am on the login page
  And the following account exists:
    | Email                  | Username    | Password |
    | maxi.muster@gmail.com  | MaxiMuster  | P4ssw0rt |
  When I select "Reset"
  And I enter my email "maxi.muster@gmail.com"
  And I submit my input
  Then I am redirected to the start page that shows the latest popular films
```

Letterboxd
CREATE ACCOUNT FILMS LISTS MEMBERS JOURNAL
Sign In
Username or Email
Password
Remember me
SIGN IN
Reset a lost or forgotten password.
About News Pro Apps Podcast Year in Review Gift Guide Help Terms API Contact
0:00 / 0:26

Does scenario 4 (Gherkin text & video) match the requirements?
If the scenario matches the requirements, please write "Matches all (Stimmt überein)". Otherwise, describe all inconsistencies that you notice.

How certain are you on answering the last question?

very uncertain ○ ○ ○ ○ ○ ○ ○ ○ very certain

FIGURE 6 One page in the questionnaire (scenario 4, experimental group).

scenario page, the participants of both groups are asked to state the inconsistencies they found (M_1, M_2), as well as their answer certainty (M_3). Figure 6 shows scenario four from the translated questionnaire (experimental group). The original instructions for the experiment were written in German and are available in our data set[27].

After checking the tenth scenario, participants are asked to answer the questions “Overall, how difficult was it to recognize inconsistencies?” (M_4, question ID *ProcEval1*[27]) and “After showing you ten scenarios: How do you assess your understanding of the functionality and usage of the presented functions?" (M_5, question ID *ProcEval2*[27]). At last, participants fill in their demographic data and assess their answer quality (seriousness, distraction, understanding of instructions). To encourage honest responses, we inform participants that their quality assessment responses have no consequences.

Participation in the experiment was rewarded because we wanted to get as many potential participants as possible. The payment for one person was 5.00 GBP for the control group and 5.50 GBP for the experimental group. The number of participants in each group was limited to eleven. We have found participants through Prolific. We set the following criteria on the platform to filter participants: residence in Germany, German as first language, and very good English skills. This way, we ensured that participants can follow German instructions, understand English text in the software, and write texts in German. The platform then sent invitations to qualified participants. Participants could conduct the experiment if they received an invitation and participation spaces were still available. The experiment was conducted on an online survey platform (https://www.soscisurvey.de).

4.2 | Demographics

For each group, 11 participants took part in the experiment. The gender distribution is almost equal (10 female, 12 male). The participants belong to different age groups, with the youngest being 19 and the oldest 60 years old. The average age is 28.7. In terms of educational level, all participants have at least a vocational baccalaureate diploma (Fachabitur in German). None of the participants have extensive knowledge of programming or BDD.

4.3 | Data Analysis

The self assessment of answer quality is adequate for checking hypotheses. According to the submitted answers, all participants have answered the questions seriously and understood the instructions. Almost all participants reported that they performed the experiment without distraction; only one reported a brief distraction.

The second author first counted metrics M_1 and M_2 for the submitted inconsistencies. Participants were asked to identify inconsistencies from demonstrated behaviors which are not in accordance with the given requirements. We count a correct inconsistency if a participant states a real difference between requirements and demonstrated behaviors. We count an incorrect inconsistency if we recognize that the participant states an inconsistency which is not related to the corresponding requirement. Then, the first author checked and corrected the counting and asked for confirmation from the second author. Only one disagreement was left after the confirmation: this was settled by consulting the third author. The exact counting reference is available in the survey data[27] (sheet "Counting Detail"). The mean and standard deviation for M_1 and M_2 are calculated between all participants.

To calculate M_3, the median value of all assessments of certainty on giving inconsistencies is calculated for each participant respectively. M_4 and M_5 are collected through answers to the mentioned questions about the difficulty of finding inconsistencies and the understanding of the development status. As these metrics are collected via Likert scale, we calculate only their median for further interpretation.

To calculate the Mann-Whitney U test for all metrics, we follow the instructions from Bortz and Schuster[28] for small samples of shared rankings. For the sake of transparency, the individual calculation steps for this test are provided in the data set[27]. According to the hypothesis definitions, all hypotheses are two-tailed. For interpretation, we set the significance level α to 0.05. Table 2 shows the statistical results.

TABLE 2 Statistical Results for All Metrics

Metric	Control Group		Experimental Group		Mann-Whitney U Test	
	Mean	SD	Mean	SD	U	p
M_1	1.91	1.64	5.73	4.88	26.5	0.024
M_2	0.36	0.81	0.36	0.50	53.5	0.557
	Median		Median		U	p
M_3	7		7		30.5	0.022
M_4	5		4		30.0	0.037
M_5	5		6		23.0	0.011

SD: Standard Deviation
Two-tailed, significance level $\alpha = 0.05$

In addition, we plot the histograms for the certainty of mentioning an inconsistency (note that this is not M_3, Fig. 7 left), the difficulty of identifying inconsistencies (M_4, Fig. 7 middle), and the understanding of development status (M_5, Fig. 7 right). We can see that participants in experimental group tend to be more certain in mentioning an inconsistency and more extensive in understanding of development status. Regarding difficulty, seven participants in experimental group find identifying inconsistency easy (4), while nine participants in control group find it difficult (5 and 6).

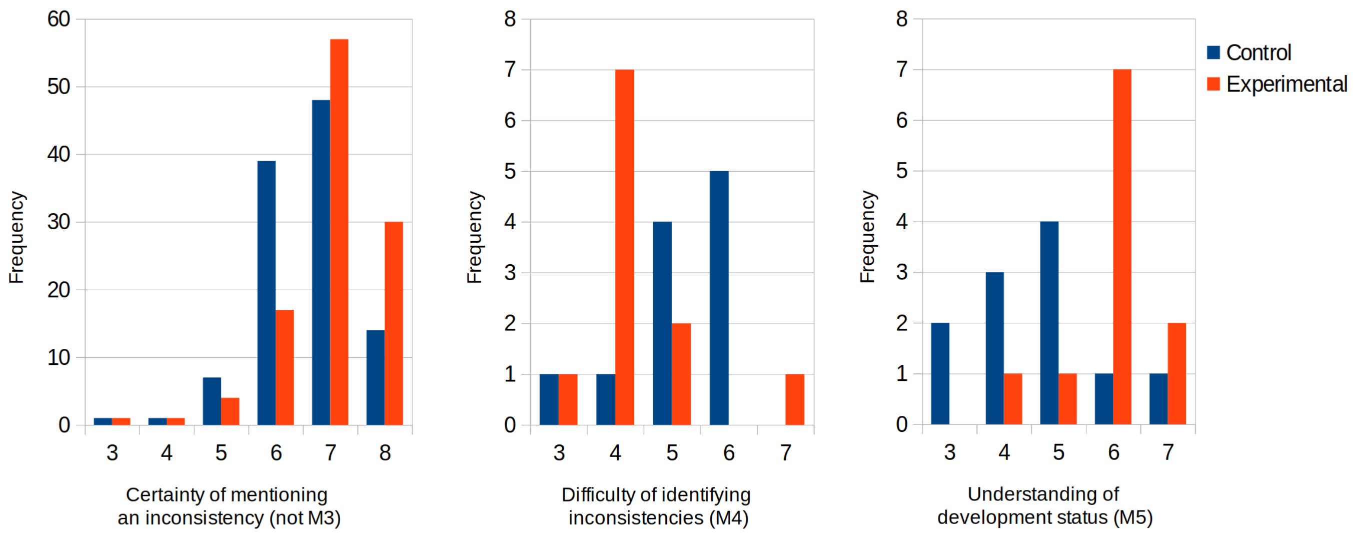


FIGURE 7 Histograms related to metrics 3, 4, and 5.

4.4 | Answers to the Evaluation Questions

Regarding M_1, a significant difference between both groups is observed (p=0.024). The mean number of correctly identified inconsistencies is higher in the experimental group than in the control group. H_1 is accepted. In contrast, results of M_2 show no significant difference (p=0.557). Most (16) participants did not falsely identify inconsistencies. Two participants in the control group stated two false inconsistencies each, while four participants in the experimental group stated one false inconsistency each. Therefore, we cannot reject the null hypothesis of H_2. Overall, we **answer *EQ*$_1$ as follows:** The difference that we have found in the mean number of correctly identified inconsistencies suggests that the GUI unit test videos and connection graph help stakeholders recognize inconsistencies with stakeholders' requirements.

For M_3, a significant difference is measured (p=0.022). H_3 is accepted. However, we cannot infer the difference direction in M_3 as the medians of both groups are the same. Results from M_4 show a significant difference (p=0.037). Participants from the control group tend to think that inconsistency recognition is difficult when viewing Gherkin specifications only (Median=5). H_4 is accepted. Regarding M_5, another significant difference is observed (p=0.011). Participants of both groups can understand the current development status. However, the experimental group has a greater median value of M_5 than the control group. H_5 is accepted. Overall, we **answer *EQ*$_2$ as follows:** The videos and connection graph of GUI unit tests appear to facilitate stakeholder feedback in two ways: stakeholders can (1) recognize inconsistencies easily and (2) understand the current development status extensively.

5 | DISCUSSION

First, we discuss some other results found in the experiment. Second, we discuss the applicability of our proposed concept in Robot Framework. Third, we imagine that Jenny obtained feedback from stakeholders through the evaluation. We show how Jenny and her team benefit from the feedback.

5.1 | Inconsistencies Found by Participants

To count M_1 and M_2, we originally used a standard solution of 24 inconsistencies. We found another 5 correct inconsistencies (shown as blue text in the survey data[27], sheet "Counting Detail") in answers from participants of the experimental group. One inconsistency is "The standard avatar is grey (instead of blue)" from participant 14 in the experimental group. Given the video, participants found inconsistencies which were not intentionally designed before conducting the experiment. This supports H_1.

5.2 | Time Cost on Viewing Videos

The length of the 10 videos used in the experiment ranges from 18 to 38 seconds. We measured session duration, i.e., how long it took a participant to complete the entire online experiment. The results show that the control group took an average of 21.0 minutes and the experimental group took an average of 31.7 minutes per person. We can infer that the participants may have used the 10.7 minutes for viewing the 10 videos. This estimated time cost is acceptable.

5.3 | Effect of the Connection Graph and Videos

In the experimental group, participants were shown the connection graph of the GUI unit tests once before the first scenario was shown. Then they were asked to identify inconsistencies for each scenario separately, while the video was shown. The correctly identified inconsistencies (M_1) should not be related to other scenarios, i.e., other GUI unit tests. Hence, we argue that mainly the video helped participants to find more correct inconsistencies (M_1).

For other metrics, the connection diagram and videos should have a combined effect. However, the effect of the connection graph itself is not separately investigated. By selecting an individual continuous GUI unit test flow, stakeholders can view a personalized video and can give feedback for a specific use case. The interaction with a connection graph and its effect on requirements elicitation is worth investigating in a future study.

5.4 | The Storage of the Videos

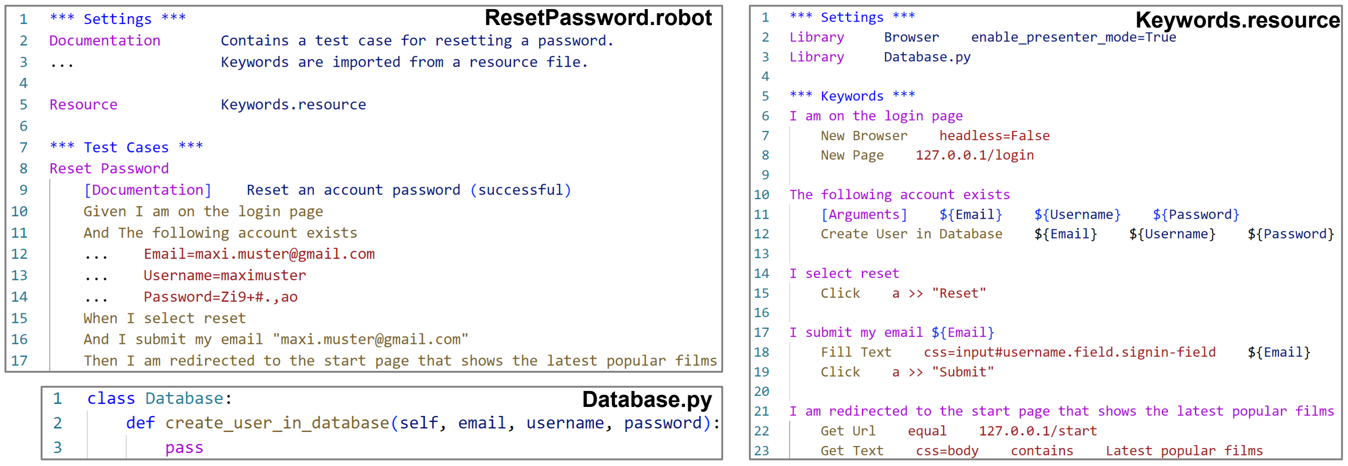
The 10 videos used in this study are approximately 7.7 MB in size. In a real project, a product owner maintains videos in the GUI Unit Test Viewer. The videos should cover all scenarios in a connection graph (as Fig. 3) and include multiple versions per each scenario. For example, if we maintain two latest versions per each scenario in Fig. 3 (10 scenarios), we would need to save 20 videos, totaling approximately 15 MB in size. This size is considerable for maintenance. The videos should be stored before the software is no longer in use. This way, stakeholders can refer to them at any time to see how the implementation of a function has evolved. If storage size becomes an issue, we recommend keeping only the last 2 to 3 latest versions of a scenario.

5.5 | Applicability in Robot Framework

Keyword-driven testing can be applied in BDD. In keyword-driven testing, a keyword defines a set of test steps and is named as a phrase (e.g. "I enter my email", line 17 in file *Keywords.resource*, Fig. 8). Daigl and Rohner[29] explain that the keywords have three levels: 1. High-level: as test steps in a test case; 2. Intermediate-level; 3. Low-level: implemented as test code or run from a person. The high-level keywords are similar to Gherkin steps, and the low-level keywords are similar to the real test code in BDD. Robot framework is one of the keyword-driven testing frameworks. Hence, we discuss how to use Robot Framework in our approach.

The test specifications in Robot Framework are maintained in files using three different data formats: robot file, resource file, and python file. The test suite is defined in a **robot** file, which consists of many test cases. Each test case consists of many keywords. The keywords are defined either in the robot file (as local keywords) or a **resource** file (as reusable keywords). The implementation code of the keywords is written in **python** files.

Daigl and Rohner[29] have shown how to write Gherkin specifications in Robot Framework. The Gherkin steps are considered as test steps in a test case. The prefixes *Given*, *When*, *Then*, *And* are ignored in Robot Framework; the following phrase is recognized as a keyword. An example is shown in Fig. 8: The keyword of the Gherkin step "Given I am on the login page" (line 10 from file *ResetPassword.robot*) is "I am on the login page" (line 6 from file *Keywords.resource*). In the definition of the keyword "I am on the login page", the login page under development is opened (line 8 from file *Keywords.resource*)).



ResetPassword.robot

```
1   *** Settings ***
2   Documentation       Contains a test case for resetting a password.
3   ...                 Keywords are imported from a resource file.
4
5   Resource            Keywords.resource
6
7   *** Test Cases ***
8   Reset Password
9       [Documentation]    Reset an account password (successful)
10      Given I am on the login page
11      And The following account exists
12      ...    Email=maxi.muster@gmail.com
13      ...    Username=maximuster
14      ...    Password=Zi9+#.,ao
15      When I select reset
16      And I submit my email "maxi.muster@gmail.com"
17      Then I am redirected to the start page that shows the latest popular films
```

Database.py

```
1   class Database:
2       def create_user_in_database(self, email, username, password):
3           pass
```

Keywords.resource

```
1   *** Settings ***
2   Library     Browser    enable_presenter_mode=True
3   Library     Database.py
4
5   *** Keywords ***
6   I am on the login page
7       New Browser    headless=False
8       New Page    127.0.0.1/login
9
10  The following account exists
11      [Arguments]    ${Email}    ${Username}    ${Password}
12      Create User in Database    ${Email}    ${Username}    ${Password}
13
14  I select reset
15      Click    a >> "Reset"
16
17  I submit my email ${Email}
18      Fill Text    css=input#username.field.signin-field    ${Email}
19      Click    a >> "Submit"
20
21  I am redirected to the start page that shows the latest popular films
22      Get Url    equal    127.0.0.1/start
23      Get Text    css=body    contains    Latest popular films
```

FIGURE 8 Example project in Robot Framework

According to private communication with test engineers at imbus AG, it is technically possible to convert feature files to robot files. Hence, we have the following suggestions of application in Robot Framework with regard to Fig. 2: At step 1, use Gherkin to communicate with stakeholders. Between step 2 and 3, convert the feature files to robot files. At step 4, develop the tests in Robot Framework; regard a test case in the robot files as a GUI unit test. At step 5, trace the time of each test step (e.g. line 11 in Fig. 8 is a step) in a test case when making a video.

5.6 | Jenny Obtains Feedback from Stakeholders

Jenny has organized an online survey with 20 stakeholders to obtain feedback. These stakeholders did not attend the kick-off meeting and have their own understanding of the requirements (as *requirements in regular sentences* in Fig. 5). After reading Gherkin-Texts with or without the connection graph and the videos, they have given their feedback for each scenario. Jenny has gathered this feedback and conducted an analysis. (We will use the results of the evaluation in this illustrative example.)

Jenny finds that this feedback indicates different types of inconsistencies:

- The related requirement has been **Incorrectly implemented**.
- The mentioned requirement is **Unimplemented**.
- The mentioned requirement has been **Additionally implemented**.
- Unclear: Further tests are needed to check if the mentioned requirement is correctly implemented.

TABLE 3 Inconsistency quotes, types, and frequency (please refer to the full list in Tab. A1 of Appendix A)

Scenario and description	Inconsistency quotes from participants (selected)	Type	Frequency
Scenario 1: Request account creation	A minimum age is required at registration.	I	1
	Successful account creation is not explicitly confirmed (via pop-up message).	U	2
Scenario 2: Log in (bad username/password)	The dark gray design of the website can be rated as “not friendly”.	I	1
	The entries will not be removed after the failed login.	U	4
Scenario 3: Log in (successful)	The user is not forwarded to start page (with popular films).	I	12
	The registration can also be done via e-mail.	Unclear	2
Scenario 4: Reset password	The user is redirected to the home page after resetting the password.	I	10
	Successful reset is not explicitly confirmed (via pop-up message).	U	5
Scenario 8: Log out	The user is not forwarded to start page (with popular films).	I	9
	Successful logging out is not explicitly confirmed (via pop-up message).	U	3
Scenario 10: Successful deactivation of account	A farewell message is displayed.	A	3
	After logging out, there is no redirection to the home page (with popular movies).	I	1

I: **Incorrectly implemented**
U: **Unimplemented**
A: **Additionally implemented**

Jenny categorized the feedback into these types and counted them, some of the results are shown in Tab. 3. Now that Jenny has a summary of the feedback results in Tab. 3, she wants to consider them in the next sprints. We list some actions that she will take. Jenny sees that the feedback "*The user is not forwarded to start page (with popular films)*" (incorrectly implemented) is mentioned about ten times for scenarios 3 and 8 each. Jenny will replace the last Gherkin steps of scenarios 3 and 8 with "And I see the start page with popular films" and set the priority of implementation to the highest possible value. For the feedback "*The entries will not be removed after the failed login*" (unimplemented) of scenario 2, four participants indicate that the entered credentials should be cleared after a failed log in. This point is concerned to be critical in IT-Security, which Jenny really cares about but did forget. She adds the Gherkin step "And I see cleared username and password fields" at the end of each scenario regarding a failed log in. For the feedback "*The registration can also be done via e-mail*" (Unclear), Jenny will adapt the Gherkin specification of Scenario 3: add another input data "maxi.muster@gmail.com | P4ssw0rt" after the Gherkin step "When I enter the following data:". She will talk to the testers about this. In addition, Jenny notices that three participants find the display of the farewell message as an inconsistency (additionally implemented). Jenny documents this point in the backlog for later discussion, because she thinks the farewell message supports the friendly impression of the web application.

5.7 | Threats to Validity

Our results are subject to some threats to validity which we discuss in the following according to Wohlin et al.[30]

5.7.1 | Construct Validity

In the online questionnaire, participants could not conduct German tasks with the software, since it was only available in English. To mitigate this threat, we set criteria on the survey platform to only select participants who have the required language skills.

One aspect that may have affected the validity of the results is that participants may have misunderstood the tasks or not read the task instructions carefully. To mitigate this threat, we conducted a pilot study with two participants and adjusted the instructions based on their feedback. In addition, we asked participants if they answered the questions seriously and if they had any points of confusion. All participants confirmed that they had answered all questions seriously and that they did not have any points of confusion. This way, we were able to minimize the participants' misunderstanding of the task instructions.

Furthermore, participants were given textual requirements which they did not collect or formulate on their own. Therefore, we could not assume that participants compared the video to the given requirements instead of their own requirements. To mitigate this threat, we presented a hint every time a new video was shown. This way, we made the given textual requirements easily accessible.

Due to the nature of an online experiment, participants might have been interrupted by their environment (e.g., phone calls, door bells, etc.). To analyze the influence of such interruptions, we investigated the quality of the given answers. As only one participant experienced one short distraction and others were not distracted according to their answers to the question "Were you able to complete the questionnaire without being distracted?" (question ID *SelfAssess2*[27]), we consider this threat to be of small relevance.

In addition, participants might have been tired which could have had an influence on their judgment. To reduce the risk of tiring the participants while doing the experiment, we described functionalities in short texts and opted for a number of ten scenarios, leading to an average duration of 26.4 min per experiment. As we assume this session duration to be adequate, we consider the loss of concentration during the experiment to have had a low influence. Nevertheless, we could not counteract tiredness caused by external factors such as participating early in the morning, or in a stressful phase.

The payment for participants may threaten the validity, because it cannot motivate participants intrinsically. Participants took part in the experiment if they accepted the invitations and free spaces were still available. We cannot influence the selection of subjects because this is dependent on the platform and the initiative of participants.

5.7.2 | Internal Validity

We used the Gherkin language to specify GUI unit tests. We must not assume that participants are familiar with this language. Therefore, it is possible that they do not understand the Gherkin text. To mitigate this threat, we performed a Gherkin training with two control questions. Participants had to answer these questions correctly to continue the experiment. Furthermore, we asked participants if they were able to follow the instructions. In case of difficulties with the given tasks of the experiment, participants were asked to communicate these difficulties at the end of the study. All participants understood the instructions according to question *SelfAssess3*[27] and have written no comments regarding any difficulties.

In the experimental group, videos may show participants functionalities that are not described in the given requirements. Participants may then write these functions as inconsistencies, which affect the collected identifications with respect to validating H_1, H_2, H_3, and H_4. To mitigate this, participants were given the following instruction: "In the videos, you may sometimes see functionalities that were not discussed in the requirements. Disregard these functionalities when looking for inconsistencies." We do not see these kinds of requirements in any submitted answers.

5.7.3 | Conclusion Validity

Our results are based on insights from 22 participants. To analyze the results objectively, we applied hypotheses testing at a significance level of $\alpha = 0.05$.

In our experiment, the videos were created manually, not from a modified GUI unit test. This might threaten the validity of answers to research and evaluation questions. To mitigate this, the second author created the video while slowly interacting with the software so that viewers can follow. In further video editing, he added rectangles to highlight every GUI interaction, as suggested by Shi and Schneider[21]. All videos are available in our data set[27].

An imprecise counting of M_1 and M_2 might be another threat. If these metrics are counted by one person, errors may happen. The second and first authors double-checked the results to mitigate this threat. The third author gave her opinion on one disagreement so that the counting was unified.

5.7.4 | External Validity

Our results must not be over-interpreted and generalized. The main limitation affecting the generalizability of our results is the lack of developer perspective. In addition, we only applied our concept in an experimental setting, with limited practical relevance. Hence, future studies are required to evaluate the benefits for the industry and the applicability in a real-world setting.

6 | CONCLUSION AND FUTURE WORK

In agile development, feedback should be elicited in short intervals and fed into development. Stakeholders who might not use the software directly can give valuable feedback. We want to involve these stakeholders by using successful GUI tests for software demonstration. We have used existing successful automated GUI tests from BDD and proposed concepts to demonstrate software under development. GUI unit tests are specified in Gherkin text according to the acceptance criteria and then implemented. In internal review meetings, a product owner can view the videos and give her feedback which can be soon considered by the development team. In an external demonstration of the software under development, stakeholders review the consolidated video and give feedback. The product owner can summarize this feedback as an orientation in planning the requirements of the next iteration. Although we propose to use videos for non-personal communication, we argue the videos can also be used in a physical meeting as a complement to live demonstration. The videos can show a software under development instantly, without the waiting time for preparing a demonstration due to technical issues.

We conducted a dedicated experiment from the perspective of the stakeholders to investigate the effect of videos in identifying inconsistencies between given requirements and demonstrated behaviors. Compared to the control group, participants in the experimental group received a connection graph and videos. Results with statistical significance have shown that (1) videos seem to help stakeholders find inconsistencies in the demonstrated software that do not match the given requirements and (2) videos and a connection graph of the videos seem to help stakeholders recognize inconsistencies easily and understand demonstrated functionalities extensively.

Based on the concept and the experiment, we answer our research question.

Answers to Research Question: The automated GUI tests are divided into several GUI unit tests, which are used to create videos. Videos are recorded in a modified test run. The test steps are explained in Gherkin while viewing the video. Stakeholders view a connection graph and videos of the GUI unit tests. This way, stakeholders can effectively give feedback to a software under development.

The effects of active selection of the connection graph and consolidated video by stakeholders can be studied in the future. In addition, we have discussed a potential application within Robot Framework and given suggested instructions. These instructions can serve as the foundation for future collaboration with industrial practitioners.

ACKNOWLEDGMENT

This work is funded by Deutsche Forschungsgemeinschaft (DFG) - Project number 289386339 (ViViUse). We thank the test engineers at imbus AG for comments about the process conception.

REFERENCES

1. Glinz M, Wieringa RJ. Stakeholders in Requirements Engineering. *IEEE Software*. 2007;24(2):18–20. doi: 10.1109/MS.2007.42
2. Bjarnason E, Wnuk K, Regnell B. Requirements are slipping through the gaps - A case study on causes & effects of communication gaps in large-scale software development. In: IEEE 2011; Trento, Italy:37–46
3. Abelein U, Paech B. State of Practice of User-Developer Communication in Large-Scale IT Projects: Results of an Expert Interview Series. In: Springer International Publishing 2014; Cham:95–111
4. Niazi M, Mahmood S, Alshayeb M, et al. Challenges of project management in global software development: A client-vendor analysis. *Information and Software Technology*. 2016;80:1–19.
5. Nicolas J, Carrillo De Gea JM, Nicolas B, Fernandez-Aleman JL, Toval A. On the Risks and Safeguards for Requirements Engineering in Global Software Development: Systematic Literature Review and Quantitative Assessment. *IEEE Access*. 2018;6:59628–59656. doi: 10.1109/ACCESS.2018.2874096
6. Groen EC, Seyff N, Ali R, et al. The Crowd in Requirements Engineering: The Landscape and Challenges. *IEEE Software*. 2017;34(2):44–52. doi: 10.1109/MS.2017.33
7. Bach PM, DeLine R, Carroll JM. Designers wanted: participation and the user experience in open source software development. In: ACM 2009; Boston MA USA:985–994
8. Paasivaara M, Durasiewicz S, Lassenius C. Distributed Agile Development: Using Scrum in a Large Project. In: IEEE 2008; Bangalore:87–95
9. Cockburn A. *Agile Software Development*. Addison Wesley, 2001.
10. Williams L, Cockburn A. Agile Software Development: It's about Feedback and Change. *Computer*. 2003;36(06):39–43. doi: 10.1109/MC.2003.1204373
11. Smart JF. *BDD in Action: Behavior-Driven Development for the whole software lifecycle*. Shelter Island, NY: Manning, 2014.
12. Shi J, Mönnich J, Klünder J, Schneider K. Using GUI Test Videos to Obtain Stakeholders' Feedback. In: IEEE 2023; Melbourne, Australia:35–45
13. Mäntylä MV, Itkonen J, Iivonen J. Who tested my software? Testing as an organizationally cross-cutting activity. *Software Quality Journal*. 2012;20(1):145–172.
14. Chawani MS, Kaasbøll J, Finken S. Stakeholder participation in the development of an electronic medical record system in Malawi. In: ACM Press 2014; Windhoek, Namibia:71–80
15. Ramesh B, Jarke M. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*. 2001;27(1):58–93. doi: 10.1109/32.895989
16. Graham D. Requirements and testing: seven missing-link myths. *IEEE Software*. 2002;19(5):15–17. doi: 10.1109/MS.2002.1032845
17. Bjarnason E, Runeson P, Borg M, et al. Challenges and practices in aligning requirements with verification and validation: a case study of six companies. *Empirical Software Engineering*. 2014;19(6):1809–1855.
18. Bjarnason E, Unterkalmsteiner M, Engström E, Borg M. An Industrial Case Study on Test Cases as Requirements. In: Springer International Publishing 2015; Cham:27–39.
19. Stangl H, Creighton O. Continuous demonstration. In: IEEE 2011; Trento:38–41
20. Pham R, Holzmann H, Schneider K, Brüggemann C. Tailoring video recording to support efficient GUI testing and debugging. *Software Quality Journal*. 2014;22(2):273–292.
21. Shi J, Schneider K. Creation of Human-friendly Videos for Debugging Automated GUI-Tests. In: Springer International Publishing 2021; Cham:141–147.
22. Tandun MA. Report-Video Production for Quick Bug-finding in the Web Applications. https://www.pi.uni-hannover.de/fileadmin/pi/se/Stud-Arbeiten/2022/BA_Tandun_2022.pdf; 2022.
23. Shi J, Schneider K, Tandun M, Karras O. Supplementary Material for Evaluating ScreenTracer among Testers. https://zenodo.org/record/7522978; 2023
24. Shi J, Karras O, Obaidi M, Tandun M. Can Videos as a By-Product of GUI Testing Help Developers Understand GUI Tests?. In: IEEE 2023; Hannover, Germany:146–153
25. Stapel K, Schneider K, Lübke D, Flohr T. Improving an Industrial Reference Process by Information Flow Analysis: A Case Study. In: Springer 2007.
26. Basili V, Rombach H. The TAME project: towards improvement-oriented software environments. *IEEE Transactions on Software Engineering*. 1988;14(6):758–773. doi: 10.1109/32.6156
27. Shi J, Mönnich J, Klünder J, Schneider K. Supporting Data Set for Paper 'Using GUI Test Videos to Obtain Stakeholders' Feedback'. https://doi.org/10.5281/zenodo.7727748; 2023.
28. Bortz J, Schuster C. *Statistik für Human- und Sozialwissenschaftler*. Springer, 2010
29. Daigl M, Rohner R. *Keyword-Driven Testing: Grundlage für effiziente Testspezifikation und Automatisierung*. Heidelberg: dpunkt.verlag, 2022.
30. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in Software Engineering*. Springer Berlin Heidelberg, 2012

How to cite this article: Shi J, Mönnich J, Klünder J, and Schneider K. Organizing GUI Tests from Behavior-Driven Development as Videos to Obtain Stakeholders' Feedback *Preprint of J of Softw Evol Proc*.

APPENDIX

A FULL LIST OF CODED INCONSISTENCY TYPES

TABLE A1 Inconsistency quotes, types, and frequency (#)

Scenario and description	Inconsistency quotes from participants	Type	#
Scenario 1: Request account creation	The kind of CAPTCHA is not specified and does not correspond to an image CAPTCHA.	I	1
	A minimum age is required at registration.	I	1
	The user is shown a welcome page after logging in (instead of start page with movies).	I	2
	Successful account creation is not explicitly confirmed (via pop-up message).	U	2
Scenario 2: Log in (bad username/password)	The registration can also be done via e-mail.	U	1
	The dark gray design of the website can be rated as "not friendly".	I	1
	The entries will not be removed after the failed login.	U	4
Scenario 3: Log in (successful)	The user is not forwarded to start page (with popular films).	I	12
	The registration can also be done via e-mail.	Unclear	2
Scenario 4: Reset password	The user is redirected to the home page after resetting the password.	I	10
	Successful reset is not explicitly confirmed (via pop-up message).	U	5
	No email was sent or it is not mentioned, that a email was sent.	U	3
Scenario 5: Add profile picture	Drag & Drop cannot be realized on a smartphone or tablet.	U	2
Scenario 6: Delete profile picture	The standard avatar is grey (instead of blue).	I	1
Scenario 7: Add favorite movie to profile	The user does not select from a poster preview.	U	5
	The number of favorite movies is not controlled. Up to 4 movies can be added.	I	4
	Successful adding of a favorite film is not explicitly confirmed (via pop-up message).	U	3
Scenario 8: Log out	The user is not forwarded to start page (with popular films).	I	9
	Successful logging out is not explicitly confirmed (via pop-up message).	U	3
	You will not have access to the profile settings after logging out, but you should also make sure that you do not have access to the rest of the profile functions (e.g. rate movies).	Unclear	1
Scenario 9: Unsuccessful deactivation of account	The incorrect entry will not be removed.	U	3
	The form says Delete instead of Deactivate.	I	2
Scenario 10: Successful deactivation of account	The form says Delete instead of Deactivate.	I	1
	A farewell message is displayed.	A	3
	After logging out, there is no redirection to the home page (with popular movies).	I	1
	Successful deactivation of an account is not explicitly confirmed (via pop-up message).	U	1

I: The related requirement has been **Incorrectly implemented**.
U: The mentioned requirement is **Unimplemented**.
A: The mentioned requirement has been **Additionally implemented**.