Using GUI Test Videos to Obtain Stakeholders' Feedback

Publisher: IEEE Cite This PDF

Jianwei Shi ; Jonas Mönnich ; Jil Klünder ; Kurt Schneider All Authors

1 Cites in Paper 150 Full Text Views

Abstract:
In software projects, stakeholders can give valuable feedback on software demonstrations. 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) tests, which show the working GUI with expected software behaviors, can be documented and then demonstrated for feedback. In our new concept, GUI tests are recorded, extended, 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. 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.

Published in: 2023 IEEE/ACM International Conference on Software and System Processes (ICSSP)

Date of Conference: 14-15 May 2023

DOI: 10.1109/ICSSP59042.2023.00014

Date Added to IEEE *Xplore*: 06 July 2023

Publisher: IEEE

ISBN Information:

Conference Location: Melbourne, Australia

Funding Agency:

SECTION I. Introduction

In software projects, the requirements of a software are defined through communication among stakeholders. Stakeholders are people or organizations who influence the requirements of a system [1]. 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 planning 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 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, extend this video with further explanations (if needed), and present this video to stakeholders who can give feedback without actually using the software themselves.

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 this paper, we make the following contributions:

1) We provide supplementary steps beside the regular agile processes to involve stakeholders in the development.

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.

3) We evaluate the effect of videos with regard to finding inconsistencies between requirements and the demonstrated software from the stakeholders' perspective.

The rest of this paper is structured as follows: Section II lists related work. The concept and its exemplary application are explained in Section III . Section IV presents the experiment conducted to evaluate the concept from the stakeholders' perspective. The collected data is analyzed and the results are discussed. The paper is concluded in Section V .

SECTION II.
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.

A. Stakeholder Participation in Testing

An industrial case study from Mäntylä et al. [11] 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. In another industrial case study from Bjarnason et al. [12], customers conducted regular tests of executable code in two companies. Another company did not involve customers until shortly before the launch of the software and therefore ran into the problem of receiving too much feedback. For effective customer testing, they [12] argue that customers need to be involved in testing at an early stage of development. Chawani et al. [13] 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.

B. Alignment of Requirements with Tests

The necessary practice of aligning requirements with tests is discussed in many studies [12], [14]–[16]. Uusitalo et al. [17] list several practices of five Finnish companies regarding the linking of requirements and testing. Bjarnason et al. [12] 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. [16] 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.

Lucassen et al. [18] explore linking user stories to test traces. Concretely, user stories are linked with methods of classes and acceptance tests. For a project of a driving assistance system, Pudlitz et al. [19] propose to annotate requirements in different levels of detail. Requirements are written in natural language and annotated with concrete elements such as signal assignments, conditions, and post-conditions. They mention that the annotation and the tracing from annotation to tests can help requirements engineers to organize testable requirements. Meanwhile, testers can develop tests that are in compliance with the requirements. Based on the given test data, requirement coverage can then be calculated to check which parts of the requirements are considered in testing.

C. Video for Demonstrating GUI Interactions

GUI Interactions can be captured in a video to communicate requirements. Mackay et al. [20] use paper drafts of a graphical software to create and use videos for finding and detailing design ideas. In a brainstorming phase, participants acted out their ideas of software usage to create videos. In a design phase, participants reviewed these videos to find one design scenario for every design problem. Paper mockups were drawn and used to show user interactions in a video. Designers watched the produced videos to understand how a user would use the software. To gain a shared understanding of requirements, Stangl and Creighton [21] propose the usage of paper GUI sketches to create videos. During video creation, clickable areas are predefined so that the prototype is executable through the video. Karras et al. [22] use digital mockups to visualize GUI interactions in a video. They propose to use digital mockups to capture and replay GUI interactions. From a developer's perspective they evaluate how videos affect comprehension in comparison to static mockups. While the control group received only static mockups, the experimental group received a video which contains interactions on these mockups. Both groups had access to textual scenario descriptions of specified GUI interactions. The training time that participants spent on reading and familiarizing themselves with the materials was measured. Ten questions were used to check comprehension. Results show that the experimental group needed significantly less training time than the control group. The number of correctly answered questions were almost the same for both groups. Karras et al. [22] conclude that textual scenario descriptions can be understood faster with the proposed video than with static mockups.

Pham et al. [23] 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 [24] follow the concept of Pham et al. [23] 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 [25] 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 was investigated among testers in a company. A semi-structured interview was conducted to collect subjective opinions of the shown videos. Shi et al. [26] 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.

The **novel contributions** in this paper are: 1. Videos are used *during* a project to collect feedback from stakeholders, not at the beginning [20] – [22]; 2. Videos are viewed not only by developers [23], [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 [21].

SECTION III.
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. Such a demonstration is not a test in itself but rather a system demonstration to obtain feedback. In agile development, it is encouraged to write tests first to ensure that the software should work as specified [27]. 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. We want to apply this concept in this agile development.

Research Question : How can existing GUI tests be used to create videos for obtaining feedback during agile development?

A. Video Production & Usage in Agile Development

Our basic concepts are:

1) Use existing GUI tests in development to create videos;
2) Demonstrate the videos to stakeholders.

Figure 1 shows how our approach supplements the general agile practices. At the beginning of agile development, requirements are elicited from stakeholders. After that, requirements are written. Then, the software and corresponding tests are programmed. We consider *Implement tests* and *Develop software* to be parallel activities, as they are not strictly ordered in agile development. Next, videos are created by using the existing tests and working software under development. In general, not only GUI tests are used, as other types of tests can also be visualized. These videos demonstrate the software to assist stakeholders in giving feedback, thus requirements are updated.

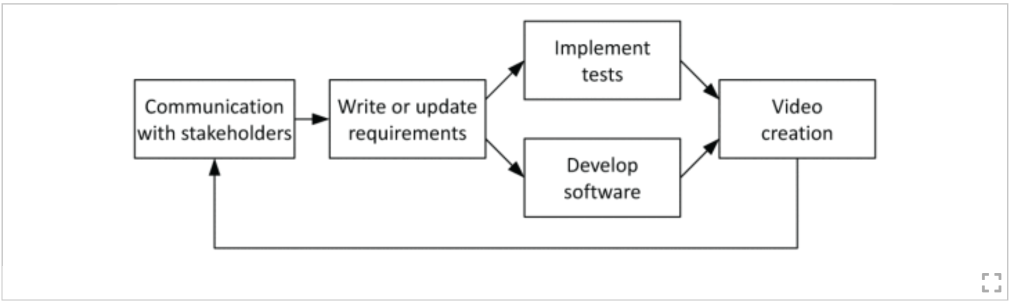
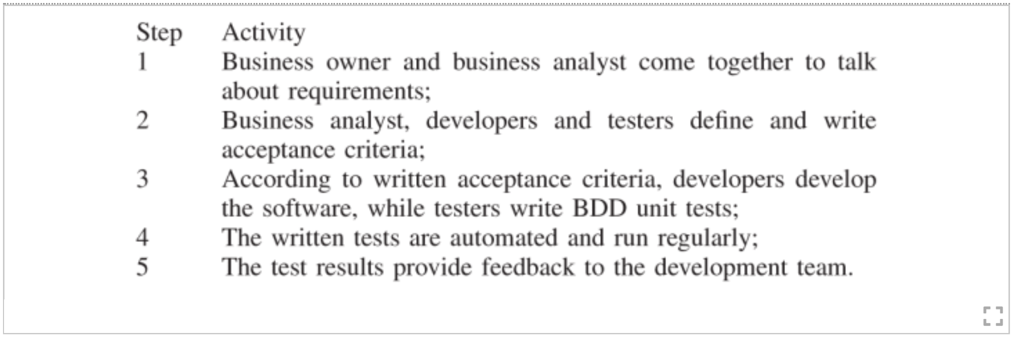


Fig. 1
Create and use video in agile development.

B. Application in Behavior-Driven Development

We apply the proposed concept in Behavior-Driven Development (BDD), an agile development method. Table I shows the concrete BDD steps according to Smart [28].

TABLE I Concrete BDD Steps According to Smart [28]



Step	Activity
1	Business owner and business analyst come together to talk about requirements;
2	Business analyst, developers and testers define and write acceptance criteria;
3	According to written acceptance criteria, developers develop the software, while testers write BDD unit tests;
4	The written tests are automated and run regularly;
5	The test results provide feedback to the development team.

Acceptance criteria contain scenarios which are specified in Gherkin. The Gherkin text 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. Step 3 (Tab. I) follows test-driven development (TDD) practice [27].

From the set of BDD tests, we use the GUI tests to create videos. These GUI tests should be unit tests. Hence, we propose the following definition:

A GUI unit test is a GUI test that interacts with logically related graphical control elements on the GUI. Figure 3 shows an example of a GUI unit test and the corresponding GUI.

1) The Supplemented BDD Processes

Figure 2 depicts the supplemented BDD processes as a FLOW diagram. We use the FLOW notation by Stapel et al. [29] . Our supplementation to the BDD steps shown in Tab. I comprises the following:

a) After step 2, the business analyst creates a connection graph that shows relationships between scenarios (i.e., GUI unit tests);

b) In step 3, GUI unit tests are implemented according to Gherkin specifications. The specifications are stored in feature files;

c) In step 4, 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 [24] ;

d) After step 4, the videos and the aforementioned graph are presented to stakeholders using our *GUI Unit Test Viewer* application. Stakeholders' feedback can be obtained during the presentation;

e) After step 5, the obtained stakeholder feedback is considered by starting again at step 1.

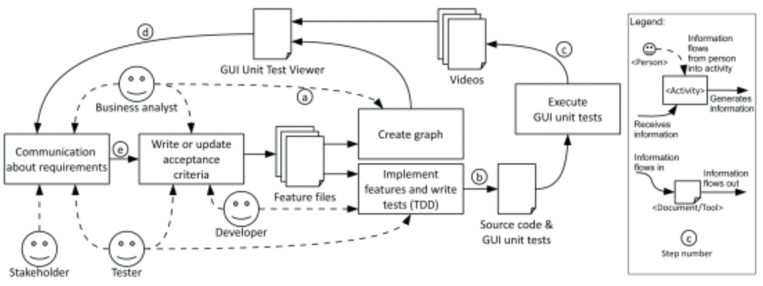
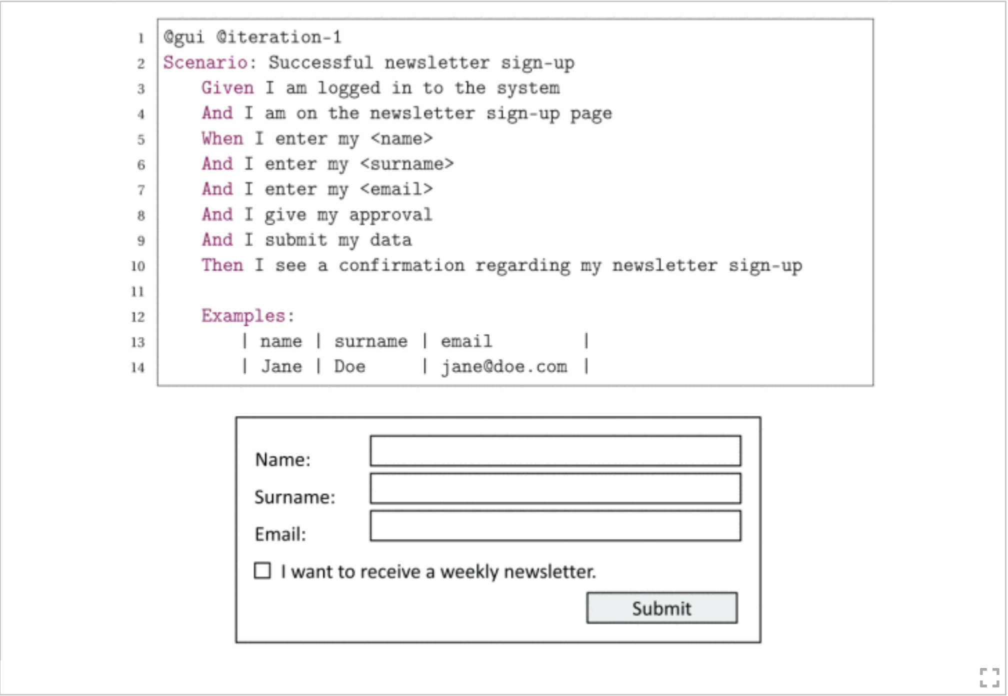


Fig. 2
FLOW diagram of the supplemented BDD processes.



```
1  @gui @iteration-1
2  Scenario: Successful newsletter sign-up
3      Given I am logged in to the system
4      And I am on the newsletter sign-up page
5      When I enter my <name>
6      And I enter my <surname>
7      And I enter my <email>
8      And I give my approval
9      And I submit my data
10     Then I see a confirmation regarding my newsletter sign-up
11
12     Examples:
13         | name | surname | email        |
14         | Jane | Doe     | jane@doe.com |
```

Name:
Surname:
Email:
I want to receive a weekly newsletter.
Submit

Fig. 3
Gherkin specification of a GUI unit test and corresponding GUI.

2) Proposed Tool Support

We have prototyped an application called *GUI Unit Test Viewer* that supports organization of GUI unit tests in a graph, presentation of videos, and documentation of feedback. Table II shows the general usage steps. In the following paragraphs, we explain these steps in detail.

In **step 1** , the Gherkin specifications of GUI unit tests are imported. The specification of a GUI unit test can be marked with the tag @gui (see Fig. 3 , line 1). Developers can also mark a GUI unit test with an iteration identifier (e.g., @iteration-1, @iteration-2, etc.). A business analyst can then recognize and import the scenarios using a tag filter.

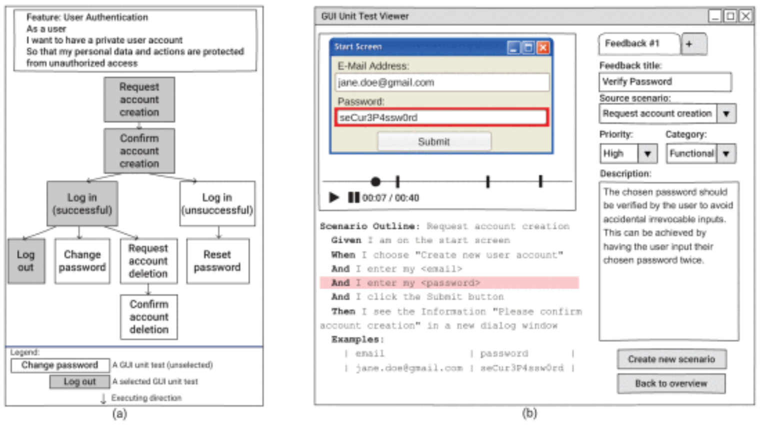
TABLE II General Usage Steps for the GUI Unit Test Viewer

Step	Description
1	Business analyst imports feature files;
2	Business analyst organizes all imported GUI unit tests in one connection graph;
3	Business analyst loads the generated videos and links them with each scenario;
4	Stakeholders select a path in the connection graph as a flow of scenarios;
5	Stakeholders see a consolidated video consisting of the selected scenarios;
6	Stakeholders give feedback about shown scenarios.

The connection graph in **step 2** shows the logical and chronological relationship between the scenarios of a feature. Figure 4a shows an example connection graph for the feature "User Authentication". This graph serves primarily as an interactive table of contents. The graph does not need to show all possible scenarios, because a stakeholder cannot view them all at once.

Prior to **step 3** , the videos are generated from GUI unit tests. The timestamps of each Gherkin step (in Fig. 3 , lines 3 to 10 are each one step) are stored as metadata together with the video. The interaction is highlighted in the test execution. The video is titled with the name of the GUI unit test. In **step 3** a business analyst uses the title information to match videos with corresponding GUI unit tests.

During demonstration, stakeholders may zoom in the graph to focus on a sub phase. In our example (Fig. 4a), the selected scenarios from **step 4** are shown as gray boxes. After a selection is confirmed, the application changes to the video view (Fig. 4b).



E-Mail Address: jane.doe@gmail.com
Password: seCur3P4ssw0rd

```
Scenario Outline: Request account creation
  Given I am on the start screen
  When I choose "Create new user account"
  And I enter my <email>
  And I enter my <password>
  And I click the Submit button
  Then I see the Information "Please confirm
account creation" in a new dialog window
  Examples:
    | email              | password       |
    | jane.doe@gmail.com | seCur3P4ssw0rd |
```

Feedback title: Verify Password
Source scenario: Request account creation
Priority: High
Category: Functional
Description: The chosen password should be verified by the user to avoid accidental irrevocable inputs. This can be achieved by having the user input their chosen password twice.

Fig. 4
Mockup of connection graph view (a) and video view (b).

Step 5 : In this view, the video and Gherkin specifications are displayed. If multiple scenarios were selected, their videos are played in succession (indicated by the bold timeline markers in Fig. 4b). Gherkin steps are highlighted during replay. The necessary synchronization is based on timestamps that have been written into the videos' metadata during video production. A consolidated video from our example is shown in Fig. 4b , where the first selected scenario (Request account creation) is played. The password field is highlighted because the "And I enter my <password>" step is currently executing.

In **step 6** , stakeholders give feedback and express new requirements for the software. Figure 4b shows a feedback panel which contains some exemplary feedback regarding a website's account creation process. Title, source scenario, priority, category and description can be specified, whereby the description can contain the problem (*what* does the customer wish to change?), the rationale (*why* does the customer wish to change it?), and a possible solution. This information is meant to primarily support the subsequent implementation process. Furthermore, after watching the video, stakeholders may have new ideas for requirements. These new requirements can be documented as Gherkin scenarios in another view by clicking "Create new scenario".

C. Technical Background of GUI Unit Tests in BDD

This subsection describes how GUI unit tests are specified and which requirements they fulfill in a BDD project. This background information supports our purposed application in BDD and serves as an orientation for industrial practitioners.

1) Position of GUI Unit Tests in a BDD Project

Figure 5 shows the different layers of a BDD project that supports supplemented BDD processes. GUI unit tests are specified at the middle level and implemented at the bottom level.

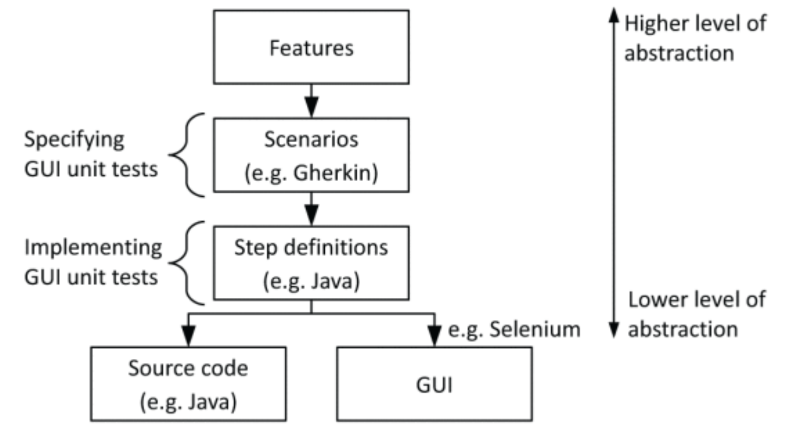

Features
Specifying GUI unit tests
Scenarios (e.g. Gherkin)
Implementing GUI unit tests
Step definitions (e.g. Java)
e.g. Selenium
Source code (e.g. Java)
GUI
Higher level of abstraction
Lower level of abstraction

Fig. 5
Layers of a BDD project. Layers can be implemented and connected through different frameworks and languages (some examples are given).

The highest, most abstract layer consists of proposed *Features* (i.e., software functionality [28]) that can be developed and deployed independently of each other. A feature is documented in a feature file, which contains title, description, and corresponding scenarios. Feature descriptions are typically documented in the same way as user stories. A number of scenarios are defined for every feature. *Scenarios* specify the behavior of the software and are used as acceptance criteria. These scenarios are specifications of GUI unit tests, e.g., Gherkin specifications. At the lowest level of abstraction, step definitions should be written in test code to implement specifications. This code tests functionalities in the source code or GUI control elements. GUI unit tests are implemented to execute steps which relate to the GUI. For example, *Selenium WebDriver* can be used to implement GUI unit tests that automate GUI interactions for web applications.

2) Position of GUI Unit Tests in the BDD Test Pyramid

In Figure 6 we present an adapted test pyramid that is based on the test pyramid by Fowler [30]. It shows a possible distribution of tests in BDD. GUI unit tests are located at the bottom, just above unit tests. Generally, the higher a test type is located in the test pyramid, the more sparingly it is used in a software project.

Unit tests are at the lowest level, which has the fastest execution speed. Unit tests should be strictly independent from each other. GUI unit tests and other automated acceptance tests are one level higher. These tests take longer to run than unit tests because it takes time to load and update the GUI. GUI unit tests are more integrated, i.e., they test several parts of the system together. In addition, they are documented in a natural language, thereby supporting communication between stakeholders. GUI unit tests can be selected and arranged in such a way that they are presentable to stakeholders as a user journey (a user's path through the system). Exploratory tests, which are conducted to test all possible combinations of functionalities, are at the highest level.

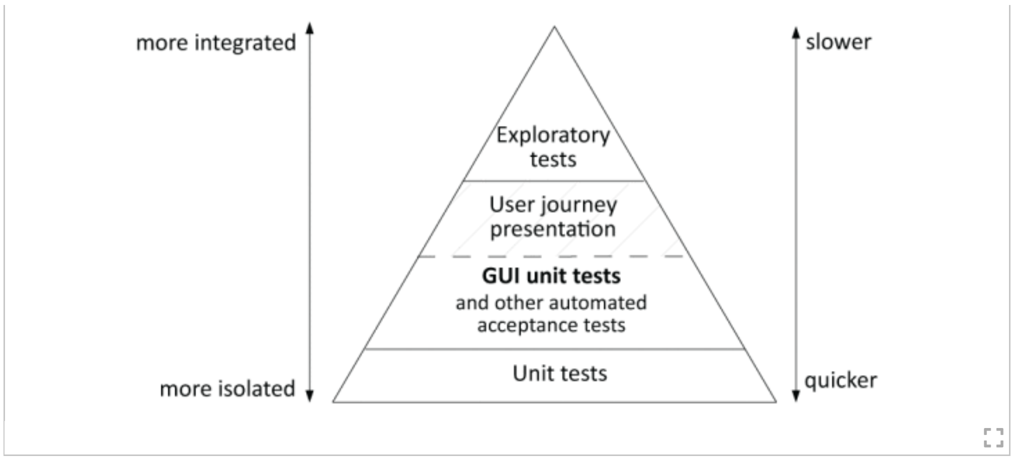


Fig. 6
The adapted test pyramid.

3) Requirements of GUI Unit Tests in a BDD Project
A GUI unit test should satisfy the following requirements in order to provide flexibility in development and clear reference in demonstration:

1) GUI unit tests are independent of each other, similar to unit tests. A GUI unit test should cover necessary system behaviors for a given program state, hence it can be executed on its own without being dependent on other test cases. Therefore, *given* steps which define preconditions should be used to set up the program state before executing the interaction. This allows developers or testers to change or delete tests without affecting the outcome of other tests.

2) A GUI unit test contains as few interaction steps as possible, since only a single GUI unit should be tested. According to the adapted test pyramid (Fig. 6), GUI unit tests are slower than regular unit tests, but fast enough to be executed regularly. A video is generated during GUI unit test execution. This way, the time cost of generating the videos is acceptable for developers and testers. The duration of the consolidated video is acceptable if a moderate number of GUI unit tests are selected for viewing.

3) For the scenario definition of a GUI unit test, an imperative documentation style can simplify the traceability between Gherkin steps and video content. Highlighting these steps and their respective GUI elements is then possible, making it easier for stakeholders to recognize and understand GUI interactions. Figure 3 shows an example of an imperative-style Gherkin text of a GUI unit test that describes a successful sign-up for a newsletter.

SECTION IV. 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 [31], 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 stakeholders

in the context of a controlled online experiment with Prolific participants in demonstrating account management functions of an online website.

A. User Study Design

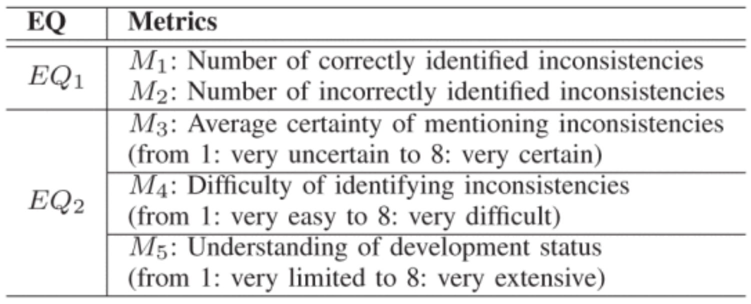
Based on the evaluation goal, we ask the following evaluation questions (EQ):

- 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. III .

Table III 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.

TABLE III Metrics for the Evaluation Questions



EQ	Metrics
EQ_1	M_1: Number of correctly identified inconsistencies M_2: Number of incorrectly identified inconsistencies
EQ_2	M_3: Average certainty of mentioning inconsistencies (from 1: very uncertain to 8: very certain)
	M_4: Difficulty of identifying inconsistencies (from 1: very easy to 8: very difficult)
	M_5: Understanding of development status (from 1: very limited to 8: very extensive)

We are interested in testing the following hypotheses:

There is a difference in

H_1: the number of correctly identified inconsistencies (M_1)

H_2: the number of incorrectly identified inconsistencies (M_2)

H_3: the average certainty of giving inconsistencies (M_3)

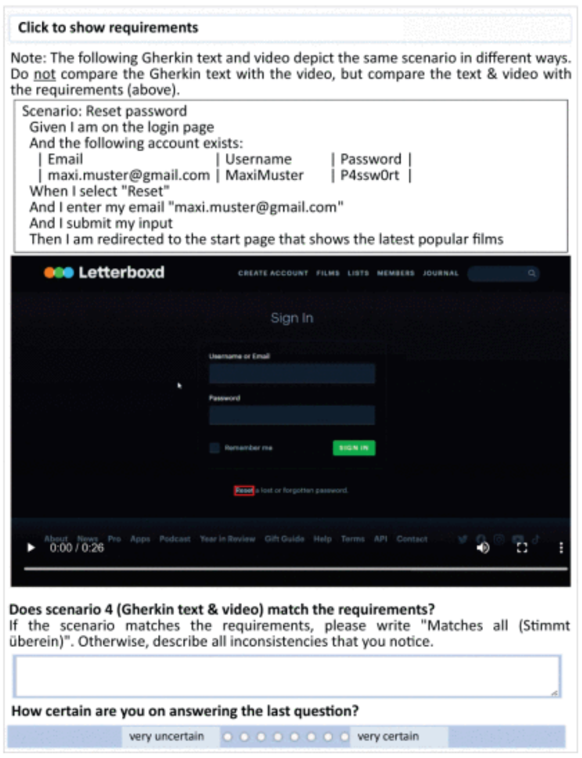
H_4: the difficulty of identifying inconsistencies (M_4)

H_5: the understanding of the development status (M_5)

between the provision of Gherkin specifications and the provision of Gherkin specifications, connection graph, and videos. The corresponding null hypotheses assume that there is no difference between both provisions in the respective metrics.

Figure 8 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. 4a). 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 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 7 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 [32].


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

Fig. 7
One page in the questionnaire (scenario 4, experimental group).

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* [32]) 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* [32]). 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.

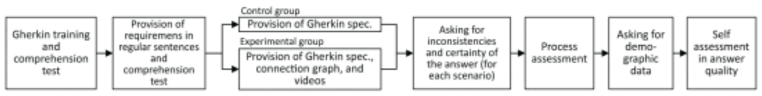

Gherkin training and comprehension test
Provision of requiremens in regular sentences and comprehension test
Control group
Provision of Gherkin spec.
Experimental group
Provision of Gherkin spec., connection graph, and videos
Asking for inconsistencies and certainty of the answer (for each scenario)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

Fig. 8
General experiment design. (Note: Spec. means Specifications)

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 an online platform. [1] 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. [2]

B. 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.

C. Data Analysis

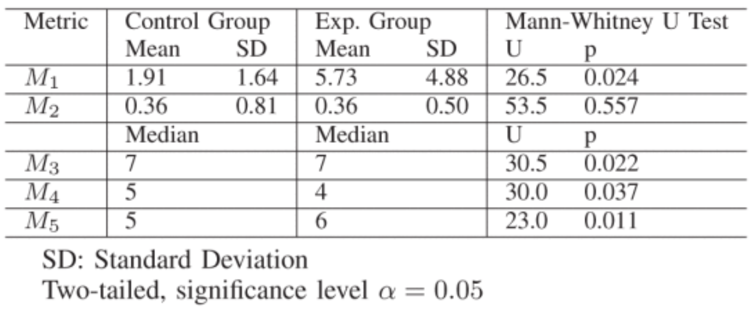
The self assessment of answer quality is adequate for checking hypotheses. According to the submitted answers, all participants have answered the questions seriously and understood the instructions. Almost all participants reported that they performed the experiment without distraction; only one reported 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 [32] (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 [33] for small samples of shared rankings. For the sake of transparency, the individual calculation steps for this test are provided in the data set [32]. According to the hypothesis definitions, all hypotheses are two-tailed. For interpretation, we set the significance level α to 0.05. Table IV shows the statistical results.

TABLE IV Statistical Results for All Metrics



Metric	Control Group		Exp. 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 $\alpha = 0.05$

D. Interpretation and Answers for 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.

E. Discussion

We discuss some other results found in the experiment and answer our research question.

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 [32], 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.

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.

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.

4) Applicability in Agile or Other Software Development

We suggest an applicability matrix (Fig. 9) that indicates the applicability of our approach. The crucial two dimensions are: 1. When are GUI tests available for video creation? (X axis) 2. How fast can the development team implement updated requirements when stakeholders provide feedback on videos? (Y axis)

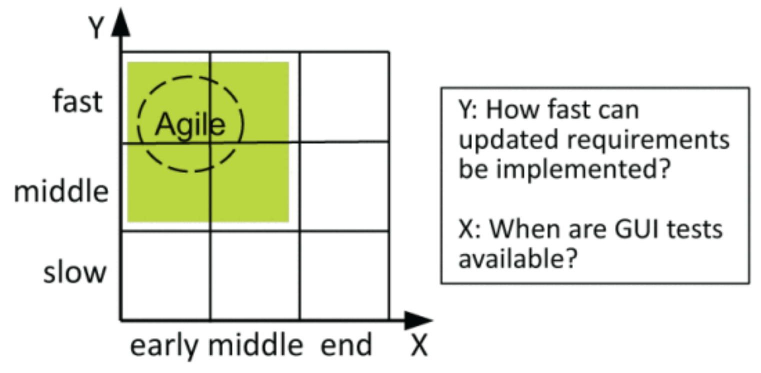

Y
fast
middle
slow
Agile
early middle end X
Y: How fast can updated requirements be implemented?
X: When are GUI tests available?

Fig. 9
Applicability matrix.

In this paper, we have proposed our concept mainly for agile development. We argue that our concept is best applicable to agile development, because (1) GUI tests can be implemented during agile development and presented as videos to obtain feedback, and (2) obtained feedback can be considered or implemented in the next iterations. In the applicability matrix (Fig. 9), agile development is depicted as a dashed circle in the upper left corner. Our concept can also be applied to development methods that are located close to agile development in the applicability matrix (green area in Fig. 9), e.g., hybrid development, where GUI tests are available in the early or middle stages of the development and changes in requirements can be considered in time.

Answer to Research Question : 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 a natural language 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.

F. Threats to Validity

Our results are subject to some threats to validity which we discuss in the following according to Wohlin et al. [34] .

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. To mitigate this threat, we conducted a pilot study with two participants and adjusted the instructions based on their feedback. This way, we were able to resolve some misunderstandings.

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* [32]), 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.

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* [32] 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.

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 [24]. All videos are available in our data set [32].

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.

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.

SECTION V.
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 GUI tests in agile development and proposed concepts to demonstrate software under development. We have applied these concepts to BDD, where acceptance criteria are written first. GUI unit tests are specified in Gherkin text according to the acceptance criteria and then implemented. By reviewing the consolidated video, stakeholders can give feedback.

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.

Our concepts can be applied in agile or hybrid development. Moreover, the use of GUI test videos could be appropriate for remote work communication between distributed stakeholders. The effects of active selection of the connection graph and consolidated video by stakeholders can be studied in the future. Furthermore, our proposed concept should be evaluated from the developer's perspective to check its feasibility and return on investment. Obtaining early feedback through GUI tests can contribute to stakeholder satisfaction and successful software projects.

ACKNOWLEDGMENT

This work is funded by Deutsche Forschungsgemeinschaft (DFG) - Project number 289386339 (ViViUse).

Authors

Figures

References

Citations

Keywords

Metrics

Footnotes