UIProtoCheck: A Checklist for Semantic Inspection of User Interface Prototypes

Conference Paper · September 2023

DOI: 10.1145/3613372.3613376

3 authors:

Gretchen Macedo
Federal University of Amazonas
7 PUBLICATIONS 35 CITATIONS
SEE PROFILE

Awdren Fontão
Federal University of Mato Grosso do Sul
87 PUBLICATIONS 404 CITATIONS
SEE PROFILE

Bruno Freitas Gadelha
Federal University of Amazonas
116 PUBLICATIONS 421 CITATIONS
SEE PROFILE

UIProtoCheck: A Checklist for Semantic Inspection of User Interface Prototypes

Gretchen T. de Macedo
Federal University of Amazonas (UFAM)
Amazonas Public Prosecution Office (MPAM)
Manaus, AM, Brazil
gtmacedo@icomp.ufam.edu.br

Awdren Fontão
Federal University of Mato Grosso do Sul (UFMS)
Campo Grande, MS, Brazil
awdren.fontao@ufms.br

Bruno Gadelha
Federal University of Amazonas (UFAM)
Manaus, AM, Brazil
bruno@icomp.ufam.edu.br

ABSTRACT

User interface prototypes are widely used in software development to facilitate customer communication and explore ideas, especially in agile development teams. In addition, they often guide subsequent stages of the development process, such as coding, testing, and training. Given their effect on the software development cycle, UI prototypes must be included in quality assurance activities such as inspections. Existing UI inspection approaches aim to detect mainly usability problems and are designed to inspect implemented software. However, development costs could be reduced by early detection of design defects if prototypes were reviewed against software requirements before implementation. For this reason, we developed UIProtoCheck, a comprehensive checklist to inspect UI prototypes semantically according to the software requirements. To evaluate it, we conducted a study where 12 participants used our checklist to inspect three UI prototypes based on a given scenario. The results showed that teams with the best results achieved 67% effectiveness in identifying semantic errors previously included in the prototypes. These promising initial results indicate that UIProtoCheck can support the semantic inspection of UI prototypes.

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**; • **Human-centered computing → User interface design**.

KEYWORDS

user interface prototyping, checklist-based reading, user interface inspection

ACM Reference Format:
Gretchen T. de Macedo, Awdren Fontão, and Bruno Gadelha. 2023. UIProtoCheck: A Checklist for Semantic Inspection of User Interface Prototypes. In *XXXVII Brazilian Symposium on Software Engineering (SBES 2023), September 25–29, 2023, Campo Grande, Brazil.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3613372.3613376

1 INTRODUCTION

User interface prototyping is a low-cost graphical model of development that defines how users will interact with software to achieve their goals [3]. Development teams broadly use this feature during requirements elicitation to facilitate communication between customers and developers and explore different solutions [9]. This is especially true in organizations that use agile development methods, where teams typically stick to developing models directly linked to requirements elicitation. In addition, UI prototypes can guide other phases of the development process, such as implementation, specification of test cases, training, and deployment [6].

In this way, it is essential to include UI prototypes in quality assurance activities since errors can propagate to the most advanced stages of the development process, resulting in increased costs related to bug fixing and rework [17]. Software inspections are one of the main methods of assuring the quality of artifacts. Such methods consist of the analysis of software design artifacts by specialists, aiming at detecting defects[23].

Different approaches to inspecting user interfaces have been developed over the last few decades. One of the most used is the Heuristic Evaluation [13], which analyzes the user interface with a set of general rules representing usability principles that user interfaces must achieve. Similar approaches focus on interface usability issues, such as perspective-based reading [16], cognitive walkthroughs[19], and formal usability inspections[10]. Another approach to UI inspection is the Semiotic Inspection Method [8]. Based on the user's interpretation of static and dynamic signals received through the interface, this approach evaluates whether it corresponds to what the application designer intended to send.

Besides usability problems, UI prototypes may also contain semantic problems, that is, design defects concerning interpreting and representing features visually according to the requirements specification. However, existing user interface inspection approaches aim to detect usability defects and improve the user experience. In addition, usability specialists conduct inspections directly on the interface of the already implemented software, which can hinder the early detection of semantic defects during the UI design.

Given the lack of base approaches for inspecting user interface prototypes focusing on semantics, this work poses the following Research Question: *How to inspect user interface prototypes for their functionality?* To address this question, we developed UIProtoCheck, a checklist for UI prototype inspection focusing on its semantic aspects. This checklist can be used during inspection routines to verify if UI prototypes represent a given feature correctly, according to a reference document describing its expected behavior.

To evaluate UIProtoCheck, we conducted a study with 12 participants. Given a scenario and our checklist, we instructed the participants to inspect three UI prototypes with four intentionally inserted defects each, focusing on their functional aspects. The team's results showed that they detected 8 out of 12 errors (67%), demonstrating that the checklist can effectively detect semantic defects in a given scenario. Though the checklist could be improved to achieve better results, it constitutes a baseline reference that can be used in comparative studies of new semantic inspection techniques for UI prototypes.

2 RELATED WORK

Several user interface inspection approaches have been proposed in the last decades. Regarding usability, the most used method is Heuristic Evaluation [13]. It is based on ten heuristics that describe the general desirable characteristics of a user interface. Based on this approach, other works were proposed containing heuristics aimed at more specific application domains. Among them, we can mention MIT2 [21], an inspection technique for mockups; HE4DWUX [24], a set of heuristics to evaluate the user experience and accessibility of websites for users who are deaf; the work of Díaz and Del Río [1], which proposed a set of heuristics to design secure and satisfying interfaces for Internet banking websites, and; the Bashir and Farooq work [2] who presented a collection of usability heuristics for smartphone apps.

Other inspection methods also focus on the usability aspects of the user interface. The Cognitive Walkthrough [22] is a learnability assessment method in which UI experts walk through the interface and check all the features to see if the user would understand what it is for and what to do to achieve its goals. A variation of this method is the Heuristic Walkthrough[15], which combines the Heuristic Evaluation and Cognitive Walkthrough, using a list of tasks that specify what should be done but not how to stimulate the UI exploration. The Perspective-based User Interface Inspection [16] is another approach that selects different aspects of software use and defines custom review procedures for them. Finally, formal usability inspections [10] review artifacts with a group of human factors experts who perform task scenarios on the interface as if they were users with different profiles. Defects found are evaluated by the group and forwarded to developers for correction.

Another approach to interface inspection is the Semiotic Inspection Method [8]. This method evaluates how the communication between the designer and an application user occurs by analyzing static and dynamic signals present in the interface. This evaluation identifies different ways of using the software and possible interpretations of its behavior. It uses documentation with exploratory and in-depth analysis and is performed in well-defined portions of the software.

Some efforts have been made to provide software support or automate the inspection of user interfaces. For example, Silva et al. [18] use static source code analysis techniques to generate state machine models of the user interface behavior so reviewers can manually compare them to the application design models. Some frameworks were also developed to support UI inspection. For example, MAUi [4] is a framework for mobile app usability inspection used to evaluate the usability of Minimal Viable Products in lean development startups. Marenkov et al. [12] also proposed a conceptual model and a framework to define usability guidelines so that UI designs can be evaluated during development.

The UI evaluation approaches listed above focus on evaluating software from the usability perspective. Even those that support task verification, as is the case with Cognitive Walkthrough [22] and Semiotic Inspection [8], or the automated UI verification approaches, are designed to evaluate already implemented software. This paper proposes UIProtoCheck as a quick alternative to inspect low-fidelity UI prototypes semantically. The development steps of our checklist are presented in the next section.

3 UIPROTOCHECK

Reading techniques are artifact review approaches that guide how reviews should be performed[20]. One such technique is Checklist-Based Reading (CBR), which guides reviewers through the inspection process based on recommendations or questions. Checklists are a list of characteristics that inform reviewers which perspectives should be considered during the inspection[7]. CBR is regarded as the baseline technique for inspecting software engineering artifacts.

Since user interface prototypes can be used at different times in the development process, including UI prototypes in quality assurance activities is essential. Therefore, we developed UIProtoCheck, a comprehensive checklist for inspecting UI prototypes focusing on the semantic aspect. It is intended for data-intensive applications, such as information systems and e-commerce. To inspect UI prototypes using this checklist, reviewers need a reference document that specifies the expected behavior of the features represented in the UI prototypes. This document can be a textual description of the system's behavior, a requirements specification, or a document describing use cases.

To create our checklist, we resorted to the traditional principles defined by Laitenberger et al. apud Chernak [5, 11] to develop checklists, a schema formed by two components: *Where to Look* and *How to Detect*. First, to define the *Where to Look* component, we analyzed the different perspectives of a UI that must be evaluated during the inspection in search of defects. We identified three perspectives: (i) the information displayed in the prototype, such as labels and pictures; (ii) data entry fields, usually related to domain attributes; and (iii) interaction elements, such as icons, links, and buttons, representing actions a user can perform in the interface.

After identifying these three perspectives, we defined how each should be analyzed (How to Detect). In this way, we created questions within each perspective to evaluate the prototype according to the Correctness, Completeness, and Consistency criteria used to assess the quality of software engineering artifacts[11]. In the context of the proposed method, these criteria were interpreted as follows:

- **Correctness** – whether the elements contained in the prototype adequately represent the functionality described in the reference document;

- **Completeness** – whether the prototype contains all elements of the represented functionality as specified in the reference document;
- **Consistency** – whether the prototype has additional elements and contradictions among its elements or between the UI prototype and the reference document.

Besides the questions related to the three perspectives, we included general questions that aim to assess whether the prototype represents clearly and unambiguously a feature described in the reference document. The resulting questions are presented in Table 1. The first part of the checklist refers to the prototype as a whole, while the remaining sections are specific to the three identified perspectives: displayed information, data entry elements, and interaction items.

4 EVALUATION

We conducted an empirical study to evaluate our checklist's effectiveness and efficiency in identifying semantic defects in user interface prototypes. In this study, participants worked in teams to inspect three UI prototypes using a given scenario and our checklist.

4.1 Preparation

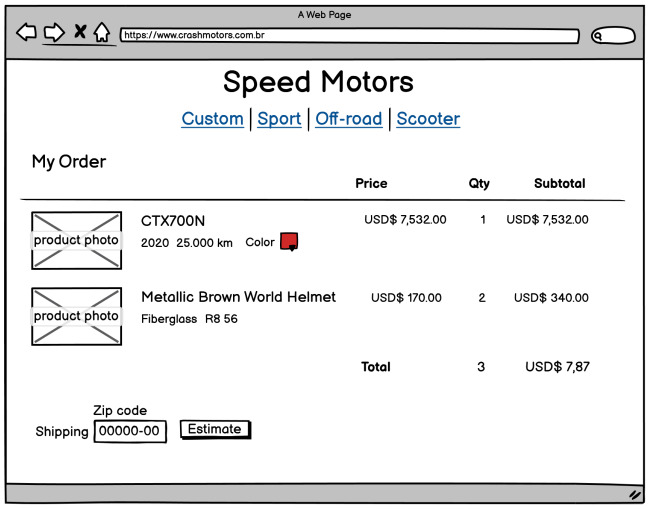
During the preparation phase of the experimental study, several artifacts needed to be created[1]. First, we prepared a consent form to be signed by students who agreed to participate in the study. Next, as we planned the study to be performed in teams to simulate a standard inspection procedure[14], we identified participants' experience in software development and quality inspections as a factor that could influence the teams' results. For this reason, we applied a questionnaire to survey the participants' professional experience one week before the study. The researchers used the results of this questionnaire to assign participants to review teams to ensure that experienced participants were distributed in all groups.

As the reference document for this study, we created a textual specification that describes an e-commerce website. The scenario presented to the participants describes an e-commerce site for motorcycles and accessories. The main features contained in the scenario are a home page with highlights and promotions, a search page, a product display page, a shopping cart, a chat, and a checkout page. We also designed three prototypes related to this scenario to be inspected during the study, corresponding to the Search, Product Page, and Shopping Cart functionalities. In total, 12 defects were inserted into the prototypes, four defects each, six related to details of the scenario and six related to the e-commerce domains. Figure 1 displays Prototype 3, Shopping Cart.

We also created a document for the participants that included the inspection process description and instructions for filling out the individual and group review forms. Other artifacts created for the experimental study include the individual and group review forms to register defects found during the experiment, a document containing the checklist, and a list of the defects we intentionally inserted into the prototypes to support the data analysis.

To review the study artifacts, we performed a pilot study with one researcher who did not participate in creating the checklist. In this study, the researcher followed the instructions document

[1]Supplementary material: https://figshare.com/s/59fcf7530732d9df48fa



		Price	Qty	Subtotal
product photo	CTX700N 2020 25.000 km Color 🟥	USD$ 7,532.00	1	USD$ 7,532.00
product photo	Metallic Brown World Helmet Fiberglass R8 56	USD$ 170.00	2	USD$ 340.00
		Total	3	USD$ 7,87

Figure 1: Prototype 3 – Shopping cart.

and inspected the prototypes using the scenario and the checklist. As a result, the researchers adjusted the study instructions, the prototypes, and the issue worksheets to facilitate the understanding of the artifacts and the manual completion during the study.

4.2 Execution

We performed the evaluation study with students in the 5th semester of a Software Engineering course. Of the 15 students who previously agreed to participate, 12 attended the study. Therefore, the researchers rearranged the teams according to the participants' experience in software engineering and formed three groups of two and two groups of three people. Table 2 displays the teams' arrangement during the experiment.

We adopted the artifacts inspection process described by Sauer et al. [14] for the study. Two stages form their process: individual preparation and meeting. In the first stage, reviewers independently review the software artifact, and in the second stage, they work as a team to consolidate their review results in a list of agreed defects.

During the study, we instructed the inspection team members to individually analyze the prototypes using the reference document and the checklist. Each reviewer annotated the issues in a review log. For each problem identified, they informed the revised prototype, the functionality represented in the prototype, a brief description of the problem, and which checklist item was violated. After individual inspection, the review team built a consolidated list of issues found by all reviewers, removing duplicate items. Finally, the review team analyzed the problems resulting from the consolidation, classifying the issues as Defect or False-Positive, and justified the given classification. We also instructed the participants to write down each review stage's start and end times.

The study execution started with presenting the inspection process and the checklist to the participants, which lasted about 20 minutes. Then, the researchers explained how to use the individual and group review forms. Finally, two researchers accompanied the study, guiding and answering participants' questions. The participants had about one hour and 40 minutes to complete the activity.

Table 1: Checklist for user interface prototypes inspection

Where to Look	#	How to Detect
UI Prototype	1	Is the represented functionality clearly identified?
	2	Does the functionality represented in the prototype correspond to a functionality specified in the reference document?
	3	Does the prototype present ambiguous information among its elements or between its components and the reference document?
	4	Does the prototype use names, terms, and descriptions according to the corresponding functionality in the reference document?
Displayed information	5	Is there consistency between the information displayed and defined in the reference document?
	6	Is there missing information in the prototype that should be displayed?
	7	Is there any additional information displayed on the prototype?
Data input fields	8	Are the prototype data entry fields consistent with the reference document (if applicable)?
	9	Should any input fields be included in the prototype?
	10	Are there additional input fields in the prototype other than those defined in the reference document?
Interaction elements (links, icons, and buttons)	11	Are the interaction elements offered by the interface prototype consistent with the functionality specified in the reference document?
	12	According to the reference document, are there any interaction elements in the interface that need to be added?
	13	According to the reference document, does the prototype offer any interaction elements other than those corresponding to its functionality?

Table 2: Participants' experience level and teams

Participant	Level of professional experience	Planned team	Study team
P1*	None	T1	–
P2	High	T1	T1
P3	None	T1	T1
P4*	High	T2	–
P5	None	T2	T2
P6	High	T2	T2
P7	None	T3	T3
P8*	High	T3	–
P9	None	T3	T3
P10	High	T4	T3
P11	None	T4	T4
P12	High	T4	T4
P13	None	T5	T5
P14	High	T5	T5
P15	Low	T5	T5

* Participants who missed the study.

4.3 Results

This section presents the analysis of the data collected during the empirical study using the checklist. We used descriptive statistics to calculate the efficiency and effectiveness of the checklist. The effectiveness was calculated from the number of defects found by each team using the following formula:

$$Effectiveness = \frac{DefectsFound}{TotalDefects}$$

DefectsFound is the number of distinct defects a team finds, and *TotalDefects* is the total number of defects intentionally inserted in the prototypes.

To calculate Efficiency, we based it on the number of defects found and the time each team used to inspect the three UI prototypes, both in the individual and group stages. Therefore, we compute the Efficiency in errors/hour using the following formula, where *TotalTime* is the time in minutes a team used to complete the whole inspection process:

$$Efficiency = 60.\frac{DefectsFound}{TotalTime}$$

To obtain the number of defects found by each team, we analyzed the teams' defect consolidation worksheets prepared in the second stage of the inspection activity. In total, participants reported 35 defects and 37 false positives. The maximum Effectiveness per team was 67% (8 out of 12 defects), and Efficiency was 6.23 defects per hour. Table 3 presents the results of defects found, false positives, effectiveness, and efficiency of each team.

The defect teams most found, with ten occurrences, were missing fields that should be included according to the specification in Prototype 1 – Search. The least common defect was the absence of a search button or icon in Prototype 1 – Search. Analyzing the results from all teams, ten of the 12 defects intentionally inserted

Table 3: Defects, false-positives, efficiency and effectiveness by team

Team	Defects	FP	Time (min)	Efficiency (errors/h)	Effectiveness
T1	8	5	82	5,85	67%
T2	5	10	88	3,41	42%
T3	7	6	88	4,77	58%
T4	7	9	96	4,38	58%
T5	8	7	77	6,23	67%

into the prototypes were reported. The two defects teams did not find are the lack of two controls: to remove items from and change items quantities in the Shopping Cart, both in Prototype 3. However, those issues were not explicitly defined as requirements in the specification. Instead, they are generally known as standard Shopping Cart features of e-commerce websites.

We also analyzed false positives qualitatively to identify why they were mistakenly reported. The qualitative analysis was performed using open coding by one researcher, and the results were revised by the other two, where they analyzed reported problems and the consolidation of defects. By false positives, we mean those defects reported by students that do not characterize semantic errors, for example, usability defects, or that refer to functionalities not represented in the prototypes included in the study. We identified five different categories, which are described in Table 4.

The first type of false positive is the incorrect identification of the feature represented in the prototype (FP1), for example, identifying the *Prototype 2 – Search* as a *Home* page. In addition, many participants filled in the *Feature* column of the individual inspection form, where was expected Search, Product Page, and Shopping cart, with terms representing more specific levels of abstraction, such as particular areas of prototypes (e.g., *"Filter motorcycle price,"*) or parts of more general features (e.g., *"Delivery address."*). This mistake led to the report of defects referring to requirements unrelated to the prototype's functionality.

Reporting missing prototypes for features not included in the study (FP2) is another common mistake. As examples, we can mention *"(P6) There is no checkout screen prototype."* and *"missing chat functionality (P2)"*. Also, it was possible to notice that the participants had difficulty differentiating usability defects, such as missing navigation links or elements' explanations, from semantic defects, where one must verify whether the prototype represents the expected behavior for the identified feature (FP3). Examples of FP3 include *"(P15) The shipping calculation field is not clearly arranged"* and *"(P15) The shipping calculation field is not clearly arranged"*.

Finally, false positives FP4 and FP5 represent dissonances between the reviewer's interpretation and the specification. For example, we can mention, "*There is no indication of the possibility of searching for parts. (P5)"*, where the participant assumed that the search only worked for motorcycles since the results presented in the prototype did not show parts.

Table 4: Categories of false-positives

ID	False-positive	Total
FP1	Incorrectly identified functionality	10
FP2	Functionality not represented in the prototypes	10
FP3	Usability problem	10
FP4	Assumed unspecified behavior	5
FP5	Illustrative text interpreted as real data	2

5 DISCUSSION

Initial results indicate that UIProtoCheck is a simple alternative for performing quick semantic inspections on early versions of interface prototypes compared to existing inspection approaches that use questions to guide UI inspections. Among such approaches, we can mention the Cognitive Walkthrough [19], the Heuristic Walkthrough [15], and the Semiotic Inspection [8]. These methods are more time-consuming because require specialists to perform detailed reviews. In addition, they are performed directly in the software, as they involve the analysis of features through the navigation.

The qualitative analysis of the reported defects indicates points that can be improved during studies with the checklist to reduce the number of false positives. The most common types of false positives, FP1, FP2, and FP3, could be mitigated with more detailed information to the participants. For example, FP1 and FP3 could be avoided with a list of features contained in the scenario and a better definition of the concepts *Functionality* and *Semantic Defects* used in the study. To avoid FP2, it could be emphasized to participants that only a few features from the specification would be represented in the prototypes.

Specifically regarding FP3, usability defects tend to be mistaken with semantic defects in user interface inspection since both act on the same artifact. However, it is worth noticing that usability problems are usually detected during inspections of more complete artifacts, where interaction details have already been thought of, while this checklist is aimed at inspecting early UI prototypes when requirements are still being mapped to possible solutions.

Although it is not the objective, reporting usability problems during semantic inspections can also anticipate detecting this type of defect or even help identify issues in the requirements specification. This is the case of false positives FP4 and FP5, which are related to participants' interpretations of the behavior and content of prototypes not specified in the reference document. These false positives indicate how much the checklist's performance in detecting defects in UI prototypes depends on the quality of the reference documents used during the inspection.

6 THREATS TO VALIDITY

Some aspects of the study can be characterized as threats to the study's validity. Firstly, the presented problem's size comprised a simplified scenario and three interface prototypes. This was necessary due to the limited time to perform the study and to obtain greater control of sample errors. Secondly, the study participants were students rather than industry professionals. To get around the experience factor, we conducted the study with 5th-period students of the Software Engineering course who already knew about software inspections. In addition, we surveyed students with professional experience and distributed them evenly in the teams.

7 CONCLUSION AND FUTURE WORK

User Interface Prototypes are helpful in many situations during the software development life cycle. As such, it is essential to include them in quality assurance activities, such as inspections, that analyze the prototype in its semantic aspect, that is, if the functionality it represents is correctly depicted. Given the lack of semantic inspection techniques for low-fidelity interface prototypes, we presented UIProtoCheck as an alternative for quick semantic inspections of low-fidelity user interface prototypes.

UIProtoCheck was evaluated through an experimental study with 12 participants who worked in teams and used UIProtoCheck to inspect three UI prototypes. The teams reached an efficiency of 6 errors per hour and an effectiveness of 67% in detecting defects previously inserted in the study prototypes. Also, the analysis of reported issues revealed factors that may have influenced many false positives. Thus, new studies must adopt measures to mitigate such factors.

Initial results indicate that UIProtoCheck can be relevant for organizations adopting agile development and must conduct quality assurance activities that require little investment in people and time. Furthermore, new inspection approaches for UI prototypes can use UIProtoCheck as a baseline method for comparison in controlled experiments.

Future work involves improving the checklist and conducting comparative studies, comparing its performance with ad-hoc inspections. Furthermore, domain-specific adaptations of the general-purpose UIProtocheck can be developed. Finally, a field study can be carried out to evaluate its practical application, involving real developers and software projects that adopt agile development and user interface prototyping.

ACKNOWLEDGMENTS

We are grateful to the Amazonas State Prosecution Office – MPE/AM for the incentive to this research provided under the Program for the Functional Improvement of Employees.

This research, carried out within the scope of the Samsung-UFAM Project for Education and Research (SUPER), according to Article 48 of Decree nº 6.008/2006(SUFRAMA), was funded by Samsung Electronics of Amazonia Ltda., under the terms of Federal Law nº 8.387/1991, through agreement 001/2020, signed with the Federal University of Amazonas and FAEPI, Brazil.

This research was also supported by the Brazilian funding agency FAPEAM through process number 062.00150/2020, the Coordination for the Improvement of Higher Education Personnel-Brazil (CAPES) financial code 001, and the São Paulo Research Foundation (FAPESP) under Grant 2020/05191-2.

REFERENCES

[1] Gloria Baños Díaz and Claudia María del Pilar Zapata Del Río. 2018. A Proposal of Usability Heuristics Oriented to E-Banking Websites. In *Design, User Experience, and Usability: Theory and Practice*, Aaron Marcus and Wentao Wang (Eds.). Springer International Publishing, Cham, 327–345.

[2] Muhammad Salman Bashir and Amjad Farooq. 2019. EUHSA: Extending Usability Heuristics for Smartphone Application. *IEEE Access* 7 (2019), 100838–100859. https://doi.org/10.1109/ACCESS.2019.2923720

[3] Elizabeth Bjarnason, Franz Lang, and Alexander Mjöberg. 2021. A Model of Software Prototyping based on a Systematic Map. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.

[4] Lin Chou Cheng. 2016. The mobile app usability inspection (MAUi) framework as a guide for minimal viable product (MVP) testing in lean development cycle. In *Proceedings of the 2nd international conference in hci and ux indonesia 2016*. 1–11.

[5] Yuri Chernak. 1996. A statistical approach to the inspection checklist formal synthesis and improvement. *IEEE Transactions on Software Engineering* 22, 12 (1996), 866–874.

[6] Gretchen T. De Macedo, Awdren Fontão, and Bruno Gadelha. 2022. Prototyping in Software Quality Assurance: A Survey With Software Practitioners. In *Proceedings of the XXI Brazilian Symposium on Software Quality*. 1–10.

[7] Rafael Maiani de Mello, Rebeca Campos Motta, and Guilherme Horta Travassos. 2016. A checklist-based inspection technique for business process models. In *Business Process Management Forum: BPM Forum 2016, Rio de Janeiro, Brazil, September 18-22, 2016, Proceedings 14*. Springer, 108–123.

[8] Clarisse Sieckenius De Souza, Carla Faria Leitão, Raquel Oliveira Prates, and Elton José Da Silva. 2006. The semiotic inspection method. In *Proceedings of VII Brazilian symposium on Human factors in computing systems*. 148–157.

[9] Steven Dow, Julie Fortuna, Dan Schwartz, Beth Altringer, Daniel Schwartz, and Scott Klemmer. 2011. Prototyping dynamics: sharing multiple designs improves exploration, group rapport, and results. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 2807–2816.

[10] Tasha Hollingsed and David G. Novick. 2007. Usability Inspection Methods after 15 Years of Research and Practice. In *Proceedings of the 25th Annual ACM International Conference on Design of Communication* (El Paso, Texas, USA) *(SIGDOC '07)*. Association for Computing Machinery, New York, NY, USA, 249–255. https://doi.org/10.1145/1297144.1297200

[11] Oliver Laitenberger, Colin Atkinson, Maud Schlich, and Khaled El Emam. 2000. An experimental comparison of reading techniques for defect detection in UML design documents. *Journal of Systems and Software* 53, 2 (2000), 183–204.

[12] Jevgeni Marenkov, Tarmo Robal, and Ahto Kalja. 2016. A study on immediate automatic usability evaluation of web application user interfaces. In *Databases and Information Systems: 12th International Baltic Conference, DB&IS 2016, Riga, Latvia, July 4-6, 2016, Proceedings 12*. Springer, 257–271.

[13] Jakob Nielsen. 1994. Heuristic evaluation. *Usability Inspection Mehods* (1994).

[14] C. Sauer, D.R. Jeffery, L. Land, and P. Yetton. 2000. The effectiveness of software development technical reviews: a behaviorally motivated program of research. *IEEE Transactions on Software Engineering* 26, 1 (2000), 1–14. https://doi.org/10.1109/32.825763

[15] Andrew Sears. 1997. Heuristic walkthroughs: Finding the problems without the noise. *International journal of human-computer interaction* 9, 3 (1997), 213–234.

[16] Forrest Shull, Ioana Rus, and Victor Basili. 2000. How perspective-based reading can improve requirements inspections. *Computer* 33, 7 (2000), 73–79.

[17] Forrest Shull and Carolyn Seaman. 2008. Inspecting the history of inspections: An example of evidence-based technology diffusion. *IEEE software* 25, 1 (2008), 88–90.

[18] J. C. Silva, J. Creissac, and J. Saraiva. 2010. GUI inspection from source code analysis. *Electronic Communications of the EASST* 33 (2010). www.scopus.com Cited By :8.

[19] Rick Spencer. 2000. The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 353–359.

[20] Thomas Thelin, Per Runeson, Claes Wohlin, Thomas Olsson, and Carina Andersson. 2004. Evaluation of usage-based reading—conclusions after three experiments. *Empirical Software Engineering* 9 (2004), 77–110.

[21] Natasha M. C. Valentim and Tayana Conte. 2014. Improving a Usability Inspection Technique Based on Quantitative and Qualitative Analysis. In *2014 Brazilian Symposium on Software Engineering*. 171–180. https://doi.org/10.1109/SBES.2014.23

[22] Cathleen Wharton. 1994. The cognitive walkthrough method: A practitioner's guide. *Usability Inspection Methods, New York* (1994), 105–140.

[23] Chauncey Wilson. 2013. *User interface inspection methods: a user-centered design method*. Newnes.

[24] Alexandros Yeratziotis and Panayiotis Zaphiris. 2018. A Heuristic Evaluation for Deaf Web User Experience (HE4DWUX). *International Journal of Human–Computer Interaction* 34, 3 (2018), 195–217. https://doi.org/10.1080/10447318.2017.1339940 arXiv:https://doi.org/10.1080/10447318.2017.1339940