Improving Model Inspection with Crowdsourcing

Dietmar Winkler[1], Marta Sabou[1], Sanja Petrovic[1], Gisele Carneiro[2], Marcos Kalinowski[2], Stefan Biffl[1]

[1]Institute of Software Technology & Interactive Systems
Vienna University of Technology, Vienna, Austria
<firstname>.<lastname>@tuwien.ac.at

Computing Institute
[2]Fluminense Federal University
Niterói, Brazil
{gcarneiro, kalinowski}@ic.uff.br

***Abstract*—Traditional Software Inspection is a well-established approach to identify defects in software artifacts and models early and efficiently. However, insufficient method and tool support hinder efficient defect detection in large software models. Recent Human Computation and Crowdsourcing processes may help to overcome this limitation by splitting complex inspection artifacts into smaller parts including a better control over defect detection tasks and increasing the scalability of inspection tasks. Therefore, we introduce a Crowdsourcing-Based Inspection (CSI) process with tool support with focus on inspection teams and the quality of defect detection. We evaluate the CSI process in a feasibility study involving 63 inspectors using the CSI process and 12 inspectors using a traditional best-practice inspection process. The CSI process was found useful by the participants. Although the preliminary results of the study were promising, the CSI process should be further investigated with typical large software engineering models.**

Keywords: crowdsourcing; model inspection; defect detection; feasibility study.

I. INTRODUCTION

Software inspection is an important and well-established task in Software Engineering (SE) to identify defects in SE artifacts early and efficiently [1]. The verification of SE models prior to the creation of software is of particular relevance for example regarding database design for business applications, creating software architectures, and agreeing on mission-critical test cases. SE model inspection typically requires checking whether a conceptual model correctly and completely represents the content of suitable reference documents, such as systems specifications [1]. Important model kinds include *Extended Entity Relationship* (EER) diagrams and UML model variants for designing databases and software system structures and behavior.

Although the verification of software models, e.g., with inspection, is key to ensure high quality for mission-critical software [4][5], it faces three important challenges. First, large software models and large associated reference documents, such as requirement specifications, are challenging to inspect with the limited resources in one inspection session of up to two hours per inspector [4]. Second, there are only limited guidelines for the cost-effective coordination of inspectors in a team to achieve systematic high-quality coverage of large inspection materials. Third, there is only limited tool support available for the coordinated inspection of software models that are aligned with inspection guidelines.

Since software model verification strongly relies on human cognitive skills, *Human Computation and Crowdsourcing* (HC&C) approaches are promising candidates for supporting human-centric software inspection. HC&C techniques rely (a) on splitting large and complex problems into many small tasks that are easy to solve for an average contributor in a suitable population and (b) on coordinating the collection and aggregation of individual micro-contributions into the overall result [2]. Therefore, HC&C techniques seem promising to (a) improve the coverage of large inspection materials by better coordination in the inspection team; (b) reduce cognitive fatigue of an inspector due to the large size of the inspection materials by selecting smaller parts suitable to find defects; and (c) speed up the inspection process by parallelizing tasks with suitable resources.

In this paper, we introduce a *Crowdsourcing-based Software Inspection* (CSI) process with tool support, evaluate the feasibility of the CSI process in a controlled experiment, and report on preliminary findings of this empirical study.

II. RELATED WORK

This section summarizes related work on Crowdsourcing in Software Engineering and on Software Inspection.

A. Crowdsourcing in Software Engineering

The notion of distributed development of software projects by large, undefined groups of contributors has been practiced in the SE community for decades, most notably within open source projects. The advent of mechanized labor ("microtasking") platforms such as *Amazon Mechanical Turk*[1] or *CrowdFlower*[2], have fueled an intensified interest in the application of crowdsourcing techniques in SE, leading to the emergence of a new research area dubbed *Crowdsourced Software Engineering* (CSE) and recently defined as *"the act of undertaking any external software engineering tasks by an undefined, potentially large group of online workers in an open call format"* [2][3].

LaToza and van der Hoek [2] distilled three models of CSE (i.e., peer production, competitions, and microtasking), depending on differentiating factors such as the contributing crowd's size (e.g., small, large), the expected time needed to

[1] Amazon Mechanical Turk: www.mturk.com
[2] CrowdFlower: www.crowdflower.com

solve each (micro)task (e.g., minutes, days), the expertise required from contributors, the incentive mechanisms used (intrinsic, extrinsic), the interdependence between tasks, or the context needed for solving each task (none, extensive).

CSE approaches have been used to solve a diversity of problems from various stages of the SE process [3]. In the planning and analysis phase, problems such as requirements acquisition, extraction and categorization can be crowdsourced. The problems from the design phase have attracted less approaches, with only a few papers attempting crowdsourced user interface and architecture design. Substantial reports focus on crowdsourcing implementation phase specific tasks such a coding and debugging. Problems that were crowdsourced from the testing phase include usability, performance and GUI testing. Within the maintenance phase, crowdsourcing was used for software adaptation, documentation and localization among others.

However, despite this diverse adoption with an intense focus on software testing and verification through crowdsourcing, employing HC&C for software model inspection has not been addressed neither in research [3] nor in practice. Our work therefore opens a new chapter in crowdsourced SE by investigating the feasibility of using crowdsourcing techniques for software model inspection.

B. Software Inspection

Software inspection [1] is a well-established formal defect detection approach that enables efficient defect detection already in early software development phases, e.g., during software design. Traditional inspection processes enable defect detection in different types of artifacts, e.g., text documents, software engineering models, or software code by following a basic three-step approach: (1) *Inspection Preparation*, where a moderator prepares the inspection package, including reference documents (e.g., requirements specifications or scenarios), inspection artifacts (e.g., software models or code), and supporting guidelines (i.e., reading techniques) [6], and team composition (team size, inspector capabilities); (2) *Individual Defect Detection* by team members and *Team Meetings* to discuss individual findings and to generate a team defect list; and (3) *Follow-up* that includes rework and inspection closure.

However, the typical two-hour slot recommended for an effective inspection session [4] limits the scope of the inspection artifact and requires appropriate scoping during inspection planning, which is very challenging for large software models and reference documents. If applicable, several inspection sessions can be scheduled to increase inspection artifact coverage. Tool support can help to reduce individual effort and improve coordination of activities and results. While tool support exists for defect detection during code inspection (e.g., *Gerrit* Code Review[3]), it is limited for model and design document inspection. Defect detection in non-software-code artifacts has been typically performed with *pen and paper* (P&P) [1]. Office suites, comprising word processors and spreadsheet solutions, can help to search, annotate, and manage individual findings with strong limitations regarding inspector coordination. Groupware tools, such as *GoogleDocs*[4], facilitate and improve inspector collaboration compared to offline office suites [7]. However, main limitations include the (a) limited capabilities to handle large artifacts such as software models, (b) limited method support for model inspection, and (c) lack of tool support for defect detection.

III. Research Issues

The main research question of this study is *whether and how HC&C mechanisms and platforms can support Software Inspection for software models*. In more detail, we aim to address two concrete research questions:

RQ.1: How can a Crowdsourcing-based Inspection (CSI) process be designed to support early and efficient defect detection? We hereby describe the proposed CSI process.

RQ.2 What are the effects of the CSI process on individual defect detection effectiveness? We present the empirical study design and preliminary findings on the effects of defect detection performance of the CSI process and a best-practice P&P inspection process approach.

IV. CSI Process Approach

The traditional software inspection approach has been extended towards a *CrowdSourcing-based Inspection* (CSI) Process (see Fig. 1 for a schematic overview of the individual process steps). Main extensions include the preparation phase (Fig. 1, Step 1) and the defect detection phase, including a *Text Analysis* task (Fig 1, Step 2a) and a *Model Analysis* task (Fig 1, Step 2b). For instrumentation we used the *CrowdFlower* (CF) platform.

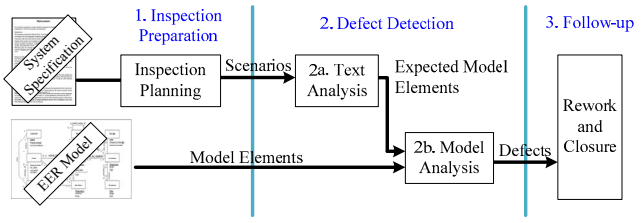

Figure 1. CrowdSourcing-based Inspection (CSI) Process.

In the preparation phase (step 1), the inspection manager splits the system specification, which is often organized into application scenarios or similar units, into a set of small entities, i.e., text fragments and sentences. These sentences can be organized as CF jobs for the *Text Analysis*, the first main process step of the CSI process. In traditional inspection, the inspector analyzes the reference document to get an overview on the concepts to check in the model in order to identify a defect. These concepts may be marked nouns that should be represented in an EER model as entities, attributes, or relationships. However, these concepts are typically not explicitly collected from the inspectors but a private intermediate result that gets discarded after inspection.

[3] Gerrit Code Review: www.gerritcodereview.com

[4] Google Docs: docs.google.com

A precondition for splitting up the traditional inspection process into small tasks that are suitable for crowdsourcing is the division between the text analysis task and the model analysis task. Therefore, the concepts that can be found in the text and should be checked for their correct representation in the model have to become explicit. We named these concepts "*Expected Model Elements*" (EMEs) since they represent this result from text analysis and are promising input to model analysis. Thus, the CSI workers have to identify these EMEs based on given requirements and scenarios, derived from the system specification. The second process step is the *Model Analysis* task, where the CSI workers receive a set of EMEs and check, whether the model correctly represents these EMEs or there are defects, which get reported in the CF application.

The inspection manager can efficiently distribute all tasks to crowd workers, who can also be recruited from an organization or a pool of experts, using the CF application. Thus, confidentiality issues can be addressed easily. The individual jobs, designed with CF are implemented as set of tasks, (a) for *Text Analysis* and (b) for *Model Analysis*. Individual results have to be aggregated at the end of the main steps for further usage. For CF support and data handling, a CF expert is required (comparable to the moderator role of an inspection process). The CSI workers represent individual inspectors (or experts) that can be recruited/invited according to their skills to support the defect detection process in the CF application.

V. Evaluation

For evaluation of the CSI process, we designed a controlled experiment (a) to investigate the feasibility of the CSI process and (b) to gain insights into the defect detection performance of a CSI process and a traditional best-practice pen-and-paper (P&P) software inspection process, i.e., the effectiveness of individual participants.

A. Experiment Description

To investigate the feasibility of the CSI process compared to a traditional inspection process with focus on individual defect detection effectiveness, we designed a controlled experiment. In this section, we summarize the study process, its design, the used artifacts, and study participants.

Study Process. The study process includes three main phases in sequential order, (a) study preparation, (b) study execution, and (c) evaluation. Study preparation includes preparing the study material (i.e., design specification, CF configurations, guidelines for task execution and questionnaires). The study execution phase includes a 30-min tutorial (i.e., an introduction for all participants) and the experiment run (with an overall duration of 120 min). The evaluation phase focuses on data collection and analysis.

Study Design. Fig. 2 presents the study design in two main groups. The first group (sub-groups A and B) applies the CSI process while the second group (group C) applies a traditional inspection process as control group. We use the control group for investigating the effects on defect detection performance with similar inspection material. For the CSI-study part, we applied a crossed design, i.e., one half of the group (sub-group A) started with *text analysis*, followed by *model analysis* while the second half of the group (sub-group B) started with *model analysis*, followed by *text analysis*. The *Expected Model Elements* (EMEs) were provided by the experiment team, based on the results of a pilot study with the CF tasks, to avoid bias between the experiment sub-groups A and B. For the CSI groups, the overall duration was split evenly between *text analysis* and *model analysis* (60 min each). Note that the *text analysis* part could be automated by using natural language processing (NPL) approaches. However, this automation tasks is out of scope in this paper.

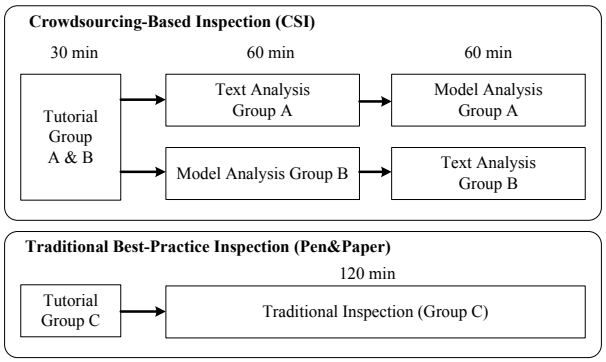

Figure 2. CSI-Study Experiment Design.

Variables. Independent variables are the study treatment and material, e.g., seeded defects of the EER model, defect types, and the CF organization; dependent variables include effort (in minutes) and effectiveness (share of reference defects found by an inspector).

Study material was a textual system specification (consisting of 3 pages, 7 scenarios, and 110 EMEs) and a medium-sized EER model (consisting of 9 entities, 13 relationships, and 32 attributes) describing basic processes of a restaurant management system. The system specification was considered to be correct, while the EER model included 33 defects, seeded by the experiment team from defects detected during the SE process, e.g., missing entities, incorrect attributes, or wrong/missing relationships and cardinalities. The control group (inspectors) received the system specification, the EER, and paper-based guidelines (as hardcopies) and were asked to report defects in a spreadsheet (which was then uploaded to the experiment server). The CSI workers received a basic guideline that described the individual CF steps for text/model analysis. For the text analysis the crowd workers received a scenario, represented by a sentence of the specification, and were asked to identify expected model elements (EMEs), attributes and relationships. For the CSI tasks, the participants received EMEs and the model, i.e., the EER model in context of this paper, and were asked to identify and report defects (i.e., missing or wrong implementations) or state that the EMEs were modeled correctly. These tasks and all other material was provided via the experiment server system (e.g., links to CF jobs) and via CF, which held the individual jobs and tasks. For *model analysis*, each inspector received up to 3 batches of 10 EMEs (out of 110 overall EMEs) linked to 3 scenarios. In the study context, the overall system specification and EER diagram were covered

by the union of individual coverage of all participants. Online questionnaires were used to collect knowledge on the background of the participants and their feedback on the individual tasks.

Study participants. The study was executed in a classroom setting with 75 participants (63 CSI and 12 P&P inspectors), who recorded their working time and any breaks. Because inspector expertise is an important issue, we captured individual experience in a questionnaire prior to the experiment. We applied a tutorial including a training session to enable participants to practice the inspection process. Furthermore, most of the students work in industrial environment in software engineering. Thus, we might consider them as junior professionals. However, in this paper we focus on the CSI process without considering individual expertise of the participants.

B. *Preliminary Results*

We analyzed the result data on defect detection performance, i.e., effectiveness of the CSI and P&P inspectors. Table 1 presents the summary of participants (per experiment group), reported candidate defects, identified true (reference) defects, and relative defect detection effectiveness. The 63 CSI inspectors reported an overall number of 934 candidate defects assigned to 411 true defects (44.0% true defect reports), while the 12 P&P inspectors reported 256 candidate defects and 120 true defects (46.9% true defect reports). On average, the CSI inspectors reported 14.8 defects (SD: 6.4) and 6.9 true defects (SD: 4.6) – this results in an average effectiveness of 20% (SD: 14%). The P&P inspectors reported on average 21.3 candidate defects (SD: 5.4) and 10 true defects (SD: 4.4) and achieved an effectiveness of 30% (SD: 13%).

TABLE I. DEFECT DETECTION PERFORMANCE

Group	Number of participants	Average reported defects	Average reported true defects	Average Effectiveness
CSI	63	14.8	6.9	20%
P&P	12	21.3	10.0	30%

Although a CSI inspector found on average less defects than a P&P inspector in the control group, it has to be mentioned that a CSI inspector saw less than 50% of the document (EME from 3 out of 7 scenarios). Therefore, this issue needs to be investigated in more detail.

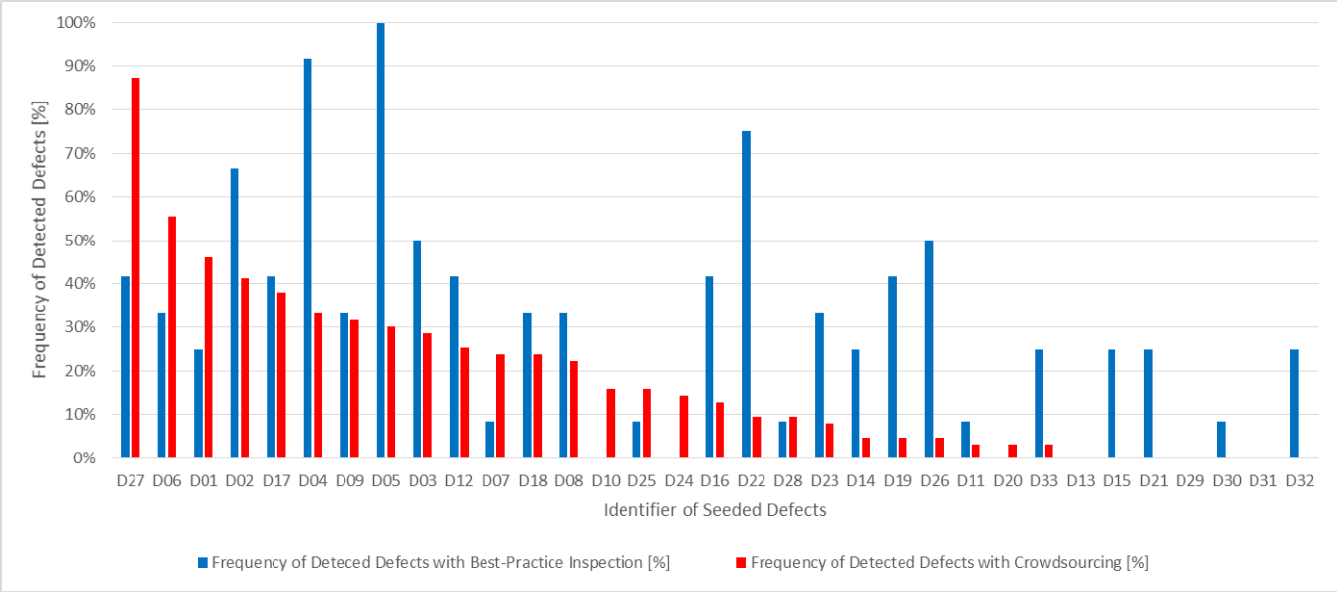

Figure 3. Frequency of Defected Defects [%] be individual inspectors in the empirical study.

Main goal of software inspection is the early and efficient identification of defects. However, an important aspect is, whether or not certain defects tend to remain undetected with a specific approach. Fig. 3 presents the initial analysis results with focus on all defects in the model (x-axis) and their detection frequency by the control and CSI groups. We observed that 6 defects were not identified by any P&P inspector and 7 defects were not detected by a CSI inspector. While 3 defects were found by CSI but not P&P, 4 defects were found by P&P but not CSI. Three defects remain unidentified both by CSI and P&P inspectors. The preliminary results indicate that a combination of traditional and CSI might be reasonable to cover a large part of the system. However, more detailed analysis and investigations are required to better understand the effects of the CSI process on defect detection performance.

C. *Threats to Validity*

To address internal validity threats, we did not allow communication between inspectors during the study execution. The overall net duration was limited to 120 min. Individual breaks were allowed with break periods reported. Prior experience of participants was captured. We applied a

random assignment of participants to the study groups. The experiment package was intensively reviewed by experts in pilot studies to identify and fix errors. Further, we executed a set of pilot runs to ensure the feasibility of the study design. To address external threats to validity, we selected a well-known application domain, a restaurant setting. We provided a tutorial to familiarize the participants with the method and technology. To avoid bias between the two CSI process steps, we used pre-defined sets of EMEs. However, there could be a possible bias in the cross-over design, because the participants in sub-group B may remember model elements when conducting text analysis. Finally, the configuration of the CF tasks has been evaluated in a set of pilot runs.

VI. Conclusion and Future Work

This paper investigated in a feasibility study to what extent the early identification of defects in software models with inspection can benefit from HC&C methods by splitting large engineering artifacts in manageable pieces of work. We introduced a *CrowdSourcing-based Inspection* process approach that can support early defect detection of large-scale engineering artifacts and models. Note that individual tasks can also be distributed to a pool of experts within an organization to address possible confidentiality issues. However, the preliminary results of an empirical study showed promising result regarding defect detection performance. However, more detailed investigations are required to clarify how CSI teams can be organized to optimize the benefits of CSI for model quality assurance and to mitigate limitations introduced with the CIS process.

References

[1] A. Aurum, H. Petersson, C. Wohlin, C., "State‐of‐the‐art: Software Inspections after 25 Years", Software Testing, Verification and Reliability, 12(3):133-154, 2002.

[2] LaToza T.D., van der Hoek A.: "Crowdsourcing in Software Engineering: Models, Motivations, and Challenges", IEEE Softw. 33(1):74-80, 2016.

[3] Mao K., Capra L., Harman M., Jia Y.: "A survey of the use of crowdsourcing in software engineering"; In: J of Systems and Software, 28p., Online; DOI: 10.1016/j.jss.2016.09.015, 2016.

[4] NASA: "Software Formal Inspection Standards", NASA-STD-8739.9, NASA, 2013.

[5] Suryn W.: "Software Quality Assurance. A Practitioner's Approach", Jon Wiley, 204p., 2014.

[6] Travassos G.H., Shull F., Carver J., Basili V.: "Reading Techniques for OO Design Inspections", Technical Report, COPPE/UFRJ-Brazil, Fraunhofer Center-Maryland, University of Maryland, 2002.

[7] Winkler D., F.J. Ekaputra, Biffl S.: "AutomationML Review Support in Multi-Disciplinary Engineering Environments", Proc. of ETFA, pp.1-8, Berlin, Germany, 2016.