Investigating Model Quality Assurance with a Distributed and Scalable Review Process

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

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

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

Abstract. [Context] Models play an important role in Software and Systems Engineering processes. Reviews are well-established methods for model quality assurance that support early and efficient defect detection. However, traditional document-based review processes have limitations with respect to the number of experts, resources, and the document size that can be applied. [Objective] In this paper, we introduce a distributed and scalable review process for model quality assurance to (a) improve defect detection effectiveness and (b) to increase review artifact coverage. [Method] We introduce the novel concept of Expected Model Elements (EMEs) as a key concept for defect detection. EMEs can be used to drive the review process. We adapt a best-practice review process to distinguish (a) between the identification of EMEs in the reference document and (b) the use of EMEs to detect defects in the model. We design and evaluate the adapted review process with a crowdsourcing tool in a feasibility study. [Results] The study results show the feasibility of the adapted review process. Further, the study showed that inspectors using the adapted review process achieved results for defect detection effectiveness, which are comparable to the performance of inspectors using a traditional inspection process, and better defect detection efficiency [Conclusions] Although the study shows promising results of the novel inspection process, future investigations should consider larger and more diverse review artifacts and the effect of the limited scope of artifact coverage for an individual inspector.

Keywords: Review, Models, Model Quality Assurance, Empirical Study, Feasibility Study.

1 Introduction

Software reviews represent important tasks in Software Engineering to identify defects in engineering artifacts early, effectively, and efficiently [3]. Formal reviews, such as software inspections [2], support software reviews for various types of engineering artifacts, e.g., written text documents, architecture diagrams, and code. The early verification of software engineering artifacts, such as software models, prior

to the construction of software code is of particular relevance for database design, software architecture, and the definition of success-critical test cases. Software model reviews typically require checking whether a conceptual model correctly and completely represents the content of suitable reference documents, such as systems specifications [2]. Example models include the *Extended Entity Relationship* (EER) diagrams or UML models to model software structures and behaviour.

Reviews for model verification face several challenges [9] regarding (a) required resources, (b) limited guidance through the review process, (c) limited document coverage for large engineering artifacts, and (d) limited tool support, as detailed next. Traditional software reviews require the availability of experts for participation in the defect detection process and team meetings. Limited availability and considerable cost make review processes challenging. Further, the typical duration of efficient reviews is limited to two hours. Thus, only a subset of the review artefact can be inspected within this time interval which limits the coverage of large and complex engineering models. Although guidelines (such as reading techniques [11]) can support the review process, it is still challenging to address the most critical system parts. Finally, typical review and inspection processes are based on *Pen & Paper* (P&P) with limited tool support that hinders coordinated reviews of software models in teams [12].

To face these challenges, we pioneer exploring how software model verification can be improved with *Human Computation and Crowdsourcing* (HC&C) methods. HC&C reduced the duration and cost of tasks that cannot be reliably automated, in fields as diverse as *Natural Language Processing* (NLP) [7], databases, or image analysis [8]. Since software model verification strongly relies on human cognitive skills, it is a good candidate for being addressed with HC&C methods. HC&C techniques rely on splitting large and complex problems into multiple, small and easy tasks solvable by an average contributor in a suitable population and then coordinating the collection and aggregation of individual micro-contributions into a larger result. Therefore, benefits for model quality assurance may include an increased coverage of large review artifacts by better coordination in the review team and accelerate the review process for large materials by parallelizing and distributing tasks with suitable resources. Furthermore, HC&C specific tools can provide tool support for model review.

The novel methodology idea has been to investigate the use of HC&C methods for software model quality assurance and model verification both at process and tool levels. First, at process (guideline) level, we propose an adapted review process for *Crowdsourcing-based Software Inspection* (CSI) to achieve faster reviews of large models by repurposing traditional software review processes. Second, at a tool support level, we explore the feasibility of implementing key review tasks within the *CrowdFlower* crowdsourcing platform[1] to perform model verification with experts. We evaluate the proposed review process in a feasibility study.

The remainder of this paper is structured as follows: Section 2 presents related work on software reviews and inspections, and crowdsourcing. Section 3 presents our key research issues. In Section 4 we describe the adapted software review process with crowdsourcing. For evaluation purposes we describe the controlled experiment in

[1] CrowdFlower: www.crowdflower.com

Section 5 and the preliminary results in Section 6. Finally, 7 concludes and summarizes future work.

2 Background and Related Work

In this section, we describe related work on Software Reviews and Inspections and Crowdsourcing in Software Engineering.

2.1 Software Reviews and Inspections

Software Reviews and Inspections are well-established and formal defect detection approaches that enable efficient defect detection already in early software development phases, e.g., during software design [2]. Traditional review and inspection processes enable defect detection with focus on different types of artifacts, e.g., text documents, graphical representations of models, or software code. Figure 1 illustrates the traditional process approach that consists of five steps: (1) Inspection Planning, where a moderator prepares the review package, including reference documents (e.g., requirements specifications), inspection artifacts (e.g., software models), and supporting guidelines; (2) Individual Defect Detection by review team members to identify defects in review artifacts according to the reference documents and by applying guidelines; (3) during a Team Meeting the inspection team generates an aggregated team defect list; (4) Rework by the author focuses on the improvement of engineering artifacts based on identified defects; and (5) Closure of the review process.

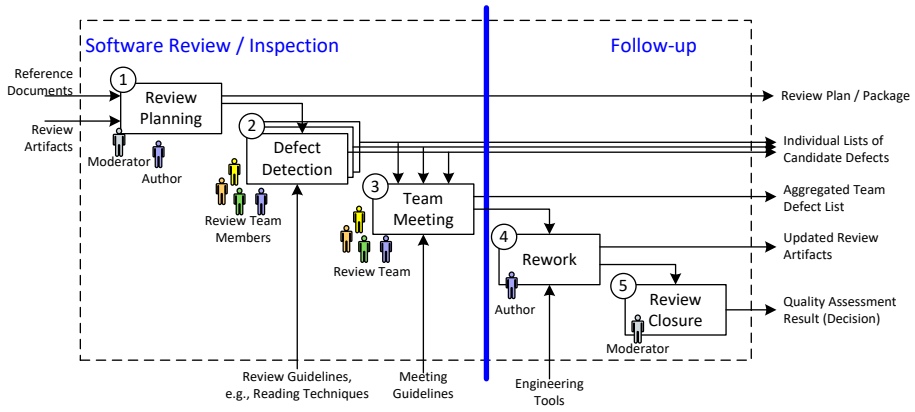

Figure 1: Best-Practice Software Inspection Approach.

Traditional software reviews and inspections are time-consuming and involve expensive experts. The overall effort for a typical inspection process depends on team size and the size of the review artifacts. Beyond preparation, coordination, and closure effort of the moderator, main effort driver focuses on individual defect detection and team meetings. Typically, two-hours are recommended for individual defect detection and team meetings. These time limits are particularly challenging for large software

models and reference documents. To address high effort for inspection, tool support can help to reduce effort and improve coordination of activities and results. While tool support exists for code reviews, it is limited for inspecting models and design documents. For instance, *CodeSurfer*[2] focuses on a fine-grained software inspection approach for software code [1]. Lessons learned from code review tools in open source development projects [9] report that commercial and open source tools, such as *Gerrit*[3], provide a web-based code review tool complemented by repository management solutions, such as GIT[4]. However, these approaches do not support inspection for non-software-code artifacts, such as design specifications or software models. Defect detection in non-software-code artifacts has been typically performed with Pen-and-Paper (P&P) [1]. Office suites, word processors, and spreadsheet solutions can support the management of individual findings but suffer from limitations regarding inspector coordination. Groupware tools, such as *GoogleDocs*[5], can facilitate and improve inspector collaboration compared to offline office suites [3].

For software model reviews, moderators and reviewers/inspectors require (a) appropriate scoping to enable efficient and effective defect detection for large-scale software artifacts and critical system parts; (b) systematic method support for defect detection, validation of defects, and coordination of inspection activities as these tasks are typically executed manually; and (c) guidelines for defect detection, such as reading techniques for model inspection.

2.2 Crowdsourcing in Software Engineering

Crowdsourcing has gained strong interest in Software Engineering (SE) and may provide promising solutions for some review and inspection issues, e.g., improved coordination (of inspection team members, tasks, and results), reduction of cognitive fatigue (by removing redundant work and reusing intermediate results from previous steps), increased coverage (as some parts of large artifacts might not be covered with traditional, weakly-coordinated approaches), more diversity (support for dealing with various inspection artifacts and access to a wider variety of inspectors), and accelerated inspection processes (by parallelization of small tasks and access to more inspectors). Therefore, we aim to explore: (a) how Human Computation and Crowdsourcing (HC&C) methods can be used to inspect SE models and (b) whether HC&C methods can lead to better model inspection by distributing and coordinating work in an inspection team.

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*[6] or *CrowdFlower*[7], have fueled an intensified interest in the application of crowdsourcing techniques in SE, leading to the

[2] CodeSurfer: www.grammatech.com/products/codesurfer
[3] Gerrit Code Review: www.gerritcodereview.com
[4] GIT: git-scm.com
[5] Google Docs: docs.google.com
[6] Mechanical Turk: www.mturk.com
[7] CrowdFlower: www.crowdflower.com

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"* [4,5].

LaToza [4] distilled three different models of CSE (i.e., peer production, competitions, microtasking), depending on differentiating factors such as the contributing crowd's size (e.g., small, large), the expected time needed to solve of 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 to extensive). In peer production models, such as those underlying open source projects, intrinsically motivated contributors (i.e., volunteers), cooperate to solve diverse interdependent tasks of a larger problem that might take several hours of weeks to solve and require an extensive understanding of the project context for being solved. Competitions style models, such as those adopted by the popular *TopCoder*[8] CSE platform, adopt a radically different approach: instead of collaboratively solving parts of a problem they elicit alternative solutions to the same problem, out of which only the most suitable solutions are selected and eventually paid for. Design related tasks where choosing from various alternatives are desired, are particularly suitable for this model. Lastly, in microtasking models, a problem is split in several, self-contained tasks, solvable in a matter of minutes by extrinsically-motivated participants with minimal expertise. This model requires a problem decomposition that leads to tasks with low interdependence and solvable with a minimal knowledge of the problem-context, thus being the most scalable thanks to the potential of intense parallelization of these task executions.

CSE approaches corresponding to the models above have been used to solve a diversity of problems from various stages of the software development life cycle [5]. In the Planning and Analysis phase, problems, such as requirements acquisition, extraction and categorization are often 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 [5] nor in practice. For example, leading software crowdsourcing platforms such as *TopCoder*[8] do not support software model verification. Our work aims to fill this gap.

3 Study Goal and Research Questions

To address the need for supporting Model Quality Assurance, in particular Model Inspection, and to improve shortcomings embodied within traditional review and

[8] TopCoder: www.topcoder.com

inspection processes, we see high potentials for introducing HC&C methods to reduce inspection resources, improve guidance for the review process, improve coordination, and to increase inspection coverage. Thus, we derived a set of research questions:

RQ.1 *How can we extend a traditional software inspection process to enable the application of* HC&C *methods?* Main goal is to design an extended inspection process that takes into consideration benefits of crowdsourcing (e.g., microtasking or coordination) applicable in traditional (adapted) software inspection processes. Thus, we introduced the concept of *Expected Model Elements* (EMEs), derived from a prior analysis step as a foundation for model quality assurance (see Section 4 for details). Further, we analyzed the adapted inspection approach, i.e., crowdsourcing-based inspection (CSI) in comparison to a traditional P&P inspection process.

RQ.2 *Does the identification of EMEs help to improve defect detection performance for model inspection?* We define the concept Expected Model Elements (EMEs) as building blocks for software engineering models, e.g., entities, attributes, and relationships, that (a) are present in requirements specifications (i.e., reference documents) and need to be modeled in the corresponding software model, e.g., in an Extended Entity Relationship (EER) diagram.

RQ.3 *What are the effects of the CSI approach with focus on (a) defect detection performance, i.e., defect detection effectiveness, and (b) coverage of the inspection artifacts?* We designed and executed a feasibility study to investigate defect detection performance and artifact coverage in a large-scale controlled experiment.

4 CrowdSourcing-based Inspection (CSI) Process

The core idea of the proposed CSI process is to split the inspection task into smaller tasks to allow parallelization of work. In the context of this paper splitting focuses on the level of the input data resources, i.e., a Text Analysis (TA) step and a Model Analysis (MA) step. Figure 2 presents the adapted inspection process that applies crowdsourcing methods to support defect detection. Main phases include (1) Review Planning; (2a) Text Analysis (TA) and (2b) Text Analysis Result Aggregation; (3a) Model Analysis and (3b) Model Analysis Result Aggregation; (4) Rework; and (5) Review Closure.

Review Planning (Step 1). Main goal of this step is to provide the technical infrastructure for crowdsourcing and the preparation of the review materials. Inputs are reference documents (i.e., a requirements specification) and a corresponding software model (e.g., EER diagrams or UML variants) to be reviewed. The requirements specification, which is often structured into application scenarios, is split into a set of small entities, e.g., text fragments or sentences by the Crowdsourcing Expert (CSE) and the moderator. Thus, each sentence represents an input for a microtask for CSI inspectors and defines the scope for the text analysis. We applied *Crowdflower* (CF) for implementation purposes and assigned a set of these microtasks to *CF-jobs* and to a set of CSI inspectors.

Text Analysis (Step 2a). For the TA, the review team members use the reference document, e.g., a requirements specification, and have to identify the main building blocks, i.e., expected model elements (EMEs), e.g., entities of the model, attributes of

model entities, and relationships between them. These building blocks are the outcome of the TA step.

Text Analysis Result Aggregation (Step 2b). As the results of the TA could also include synonyms, if they appear in the reference document, singular or plural, or typos, these deviations have to be adjusted and aggregated. Output of this step is an agreed and aggregated list of EMEs provided by a crowdsourcing expert or the moderator that represents the input for the MA step.

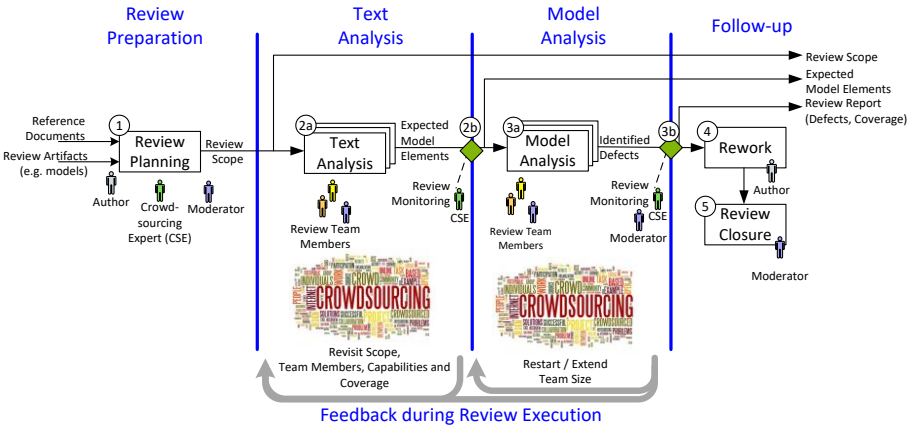

Figure 2: CrowdSourcing-based Inspection (CSI) Process.

Model Analysis (Step 3a). During the MA step the CSI inspectors apply individual EMEs and analyze given models, e.g., an EER diagram, to (a) find out whether or not the EME is present and (b) if the EME has been modeled correctly. Candidate defects have to be reported.

Model Analysis Results Aggregation (Step 3b). Similar to the TA results, MA results have to be adjusted and aggregated to an agreed list of candidate defects. This aggregated list of defects represents the input for follow-up review phases. Note that process steps 4 and 5 are similar to traditional inspection processes.

For the implementation of the CSI process we used *CrowdFlower* and a compiled a set of sentences to a *CF job* for the TA and a set of EMEs for the MA. Note that this setting enabled the inspection management (crowdsourcing expert and/or moderator) to (a) balance the work load for the CSI inspectors and (b) flexibly include additional CSI inspectors if further analysis results are required for assessing and adjusting the results of the individual steps (e.g., not enough or conflicting judgements). Based on the distributed setting of the CSI process resource issues (e.g., availability of experts) can be addressed easily. Note that the CSI inspectors represent individual inspectors (or experts) that can be recruited/invited to support the defect detection process, driven by the CF application.

5 Feasibility Study Description

To investigate the effects of the CSI process, we conducted a feasibility study. This section summarizes the study description, i.e., study process and variables,

experimental setup, participants, study material, and threats to validity. We used this controlled experiment [13] to investigate the effect of the CSI process compared with a traditional P&P inspection process.

5.1 Study Process and Variables

The *study process* consists of study preparation, execution, and data analysis. Study preparation includes the preparation of the material for CSI and the traditional software inspection approach (reference documents and scenarios, guidelines, list of reference defects, and questionnaires), the setup of the controlled experiment (tool setup for CSI and traditional inspection, study group definition, and schedule), and pilot runs. The study execution phase includes tutorials for CSI and inspection, and the experiment execution. Data analysis focuses on preliminary data screening, assignment of reported defects to reference defects, and evaluation of research questions. This paper reports on preliminary findings of the study after the first data analysis run.

In the study context we used *dependent and independent variables*: Independent variables include the seeded defects of the software (EER) model, defect types, tool configuration, and the study treatments (detailed in the next subsection). Dependent variables include effort for task execution (in minutes), reported and true defects, effectiveness (share of reference defects found by a participant), and efficiency (reference defects found per time interval, i.e., per hour).

5.2 Experimental Setup

The study design consists of two main groups (Figure 3 presents the basic experiment setup). The first group (sub-group A and B) adopts the CSI approach and the second group (sub-group C) uses the traditional best-practice inspection process and therefore plays the role of a control group. Common to all study groups is a tutorial (30 min) related to the method applied including a small practical example to get familiar with methods under investigation and related tool support.

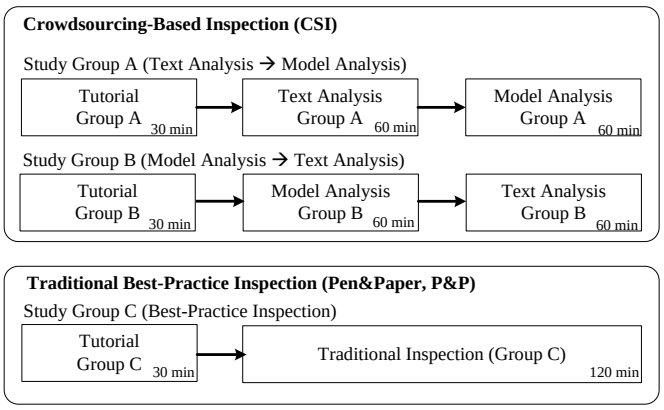

Figure 3: Setup of the Controlled Experiment.

We applied a cross-over design for the CSI part of the study, i.e., text analysis (60 min) followed by the model analysis (60 min) for group A and similar tasks in an inverse order for group B. Group C applied a traditional best-practice software inspection (120 min). For the model analysis we used a pre-defined set of Expected Model Elements (EMEs) to avoid dependencies between different tasks within the experiment groups. Note that these EMEs were provided by the experiment team, i.e., the authors. We used different experimental material for the tutorials (application domain: parking garage scenarios) and the experiment (application domain: restaurant scenarios).

5.3 Subjects and Population

The study was an integral part of a university course on "Software Quality Assurance" with undergraduate students at Vienna University of Technology. We applied a classroom setting with an overall number of 75 participants. The group assignment was based on a random distribution of participants to study groups. Because we consider group C as a control group we assigned more participants to groups A and B. During the experiment we had 63 CSI and 12 P&P inspectors.

5.4 Study Material and Tools

We applied a well-known application domain, i.e., typical scenarios and processes of a restaurant to avoid domain-specific knowledge limitations. Study material was a *textual reference document*, i.e., a system requirements specification including 3 pages in English language, consisting of 7 scenarios, and mentioning approximately 110 Expected Model Elements (EMEs). All 33 sentences of the requirement specification were numbered as vehicle for defect reporting and referring purposes. The system requirements specification was considered to be correct. For *model inspection* we used a medium-scale Extended-Entity Relationship (EER) Diagram including 33 seeded defects. These seeded defects were introduced by the experiment team (i.e., the authors) based on defects detected during the software engineering process. Furthermore, we used an experience questionnaire to capture the background skills of the participants and feedback questionnaires after each step of experiment process. Finally, we provided guidelines that drove the experiment and the inspection process.

Material that was provided to the control group (P&P inspectors) included the system specification, the EER, and guidelines as hardcopies. The CSI inspectors received a printed version of the guidelines. These guidelines (for P&P and CSI) were also available via our Experiment Management System (EMS) holding all relevant information sets. We used the following tool set:

- *Google.forms* were used for capturing the experience of participants and feedback after finalizing individual tasks, i.e., text analysis, model analysis (group A and B), or software inspection (group C).
- A *spreadsheet solution* has been used by the P&P inspectors (group C) to capture individual defect reports.

- The *CrowdFlower* application has been used to drive the text and model analysis task for the CSI inspectors. For *model analysis*, each inspector received up to 3 batches of 10 EMEs (out of 110 overall EMEs) linked to 3 scenarios. These batches of tasks have been assigned to the CSI participants via an *Experiment Management System (EMS)*.
- An *Experiment Management System (EMS)* has been used to guide the participants through the individual experiment steps.

5.5 Threats to Validity

In this section, we identify and discuss the potential threats to validity of our study and describe how we addressed them.

Participants were 75 undergraduate students of computer science and business informatics at the Vienna University of Technology. The study was a mandatory part of the course on "Software Quality Assurance". Most of the participants work at least part-time in software engineering organizations. Thus, we consider them as junior professionals comparable to industrial settings. We used an experience questionnaire to capture and assess prior experiences and skills. *Application domain.* We used typical scenarios and requirements derived from restaurant processes. Thus, all participants are familiar with this application domain.

Group assignment. We applied a random distribution of the group assignment using a sort card algorithm. We provided a tutorial to overcome method and technological limitations.

Study preparation. The experiment team (i.e. the authors) introduced 33 reference defect in the EER diagram based on typical defects collected during typical software engineering processes. However, in this paper we report on preliminary findings of the study after the initial mapping of reported candidate defects and seeded defects. During the analysis additional defects might come up and will be considered in follow-up data analysis steps (out of scope of this paper). The *experiment package* was intensively reviewed by experts to avoid errors. Furthermore we executed a set of pilot runs to ensure the feasibility of the study design.

Study execution. To address internal validity we avoided communication of the individuals during the study execution phase. The overall duration was limited to 120 min. However, individual breaks were allowed; break periods had to be reported. To avoid bias between the two CSI process steps we used pre-defined set of EMEs. However, there could be a possible bias in the cross-over design because the participants are aware of the EER model after the model analysis phase of the experiment. For the model analysis we used a pre-defined set of Expected Model Elements (EMEs) to avoid dependencies between different tasks within the experiment groups. These EMEs were provided by the experiment team, i.e., the authors.

6 Preliminary Results

This section summarizes the preliminary findings of the feasibility study with focus on effort, defect detection effectiveness, and defect detection efficiency.

6.1 Effort

We calculated the defect detection effort based on the reported starting and end time for the P&P and CSI inspectors. The CSI defect detection task is split into text analysis and model analysis tasks. Because the text analysis (i.e., identification of EMEs) is not directly related to defect detection for CSI, we added 60 min (assigned to the text analysis step) to the model analysis duration. Table 1 presents the duration of the tasks.

Table 1: Duration of P&P and CSI Tasks [in min].

Group	Number of participants	Mean	Std.Dev	Min	Max
CSI	63	113 min	11.8 min	87 min	140 min
P&P	12	107 min	26.3 min	28 min	135 min

Note that we set an upper limit of 120 min for P&P and 60 min for CSI (plus 60 min for another task, text analysis). The results showed that P&P requires on average less time for defect detection but included a higher standard deviation. We also identified one P&P inspector that had to leave earlier and spent only 28 min for defect detection. Although the upper time limit was set to 120 min, some inspectors required more time to complete the P&P task / CSI task. The initial results showed a comparable effort spent for defect detection.

6.2 Effectiveness

The main task of both study groups was to identify defects and report candidate defects. Table 2 presents the preliminary results of the reported candidate and true defects (i.e., reported defects that were matched to a reference defect). If more than one reported defect corresponded to the same reference defect, this defect was only counted once at the first time of detection.

Table 2: Reported Candidate Defects / True Defects.

		Reported Defects		Reported True Defects	
Group	Number of participants	Mean	Std.Dev	Mean	Std.Dev
CSI	63	14.8	6.42	6.9	4.62
P&P	12	21.3	5.42	10.0	4.40

We observed a higher number of reported candidate and true defects for the P&P group compared to the CSI group. Nevertheless, the CSI group spent at most one hour for defect detection. A more detailed analysis is necessary to normalize these findings.

Based on the identified true defects, effectiveness refers to the share of true defects found. The EER diagram includes 33 true defects, seeded by the experiment team. Table 3 presents the descriptive statistics.

Table 3: Defect Detection Effectiveness [%]

Group	Number of participants	Mean	Std.Dev	Min	Max
CSI	63	20%	14%	0%	67%
P&P	12	30%	13%	12%	52%

On average, the observation showed advantages for the traditional P&P approach, 30% (P&P) versus 20% (CSI). Although the CSI study group includes 3 participants who did not identify any true defects, we observed 2 participants who outperformed the P&P group. The defect detection time for CSI is limited to 60 min while the P&P inspectors worked for 120 mins. Following these observations, more detailed investigations of the preliminary results are required to better understand the effects of CSI and P&P on defect detection effectiveness.

6.3 Efficiency

Defect Detection Efficiency refers to true defects found per time interval, i.e., per hour. Table 4 presents the preliminary results of defect detection efficiency for CSI and P&P.

Table 4: Defect Detection Efficiency [Defects per hour]

Group	Number of participants	Mean	Std.Dev	Min	Max
CSI	63	6.7	4.8	0	23
P&P	12	5.7	2.1	2.4	9

The results showed advantages for CSI participants: they identified on average 6.7 defects per hour compared to P&P inspectors (5.7 defects per hour). We also identified one CSI inspector who reported 32 defects (thereof 22 true defects) resulting in an effectiveness of 67% and an efficient value of 23 defects per hour (this particular inspector required less than an hour to complete his work). On the other hand, we also identified a set of CSI inspectors who did not find any true defects (in contrast, every P&P inspectors identified at least one true defect). Further investigations are needed to identify the rationale behind these findings. However, the preliminary observations tend to support our expectations that CSI can support defect detection with crowdsourcing techniques.

6.4 CSI Process Observations

The experiment has been conducted in a class-room setting. Thus, the experiment team was able to observe the experiment process and the defect detection approach applied by the participants. Furthermore, benefits and limitations of the CSI process

approach has been discussed with industry and research experts. Table 5 summarizes the most critical process observation results for the needs of software inspection improvement for large and complex software models.

Table 5: CSI Process Observations.

Requirement	P&P Inspection	CSI
Required Resources	Co-Located	Distributed
Review Experience	Medium/High	Low/Medium
Defect Detection Guidance	Given by Reading Technique	Driven by EMEs.
Document Coverage	Low within a 2 hours interval	High, given by the task distribution
Scalability	Limited by resources	Scalable by extending the number of (judgements of) crowd inspectors
Tool Support	Limited for model reviews	Application of crowdsourcing platforms.

The application of crowdsourcing platforms such as CF enables the distribution of derived "micro tasks" among a group of experts and/or CSI inspectors. Thus, co-located reviews and inspections do not represent a limiting factor. Small tasks also support less and medium experienced inspectors that are guided by the configuration of the micro tasks. However, experienced inspectors might be required if a comprehensive view on the overall system is required. For traditional inspections, where the typical review duration is scheduled for 2 hours of working time, the coverage of these reviews is limited to available resources. Achieving high coverage for large and complex software models is challenging and might require high coordination effort between various (manual) inspection activities. In contrast to traditional inspection, the CSI process approach scales up also for large and complex software models because it depends on the configuration of the CSI jobs and tasks and the configuration of the inspection process. Thus, the document coverage can be ensured / increased by adding more CSI inspectors or adding additional judgments in case of limited quality. Finally, limited tool support for model reviews hinder efficient P&P inspection while for the CSI process approach appropriate CrowdSourcing platforms, such as CrowdFlower, can be used to support CSI inspection.

7 Conclusion and Future Work

In this paper we introduced a *CrowdSourcing-based* Inspection (CSI) process to support early defect detection of large-scale software engineering artifacts and models. The CSI process is based on a traditional inspection process by splitting up software inspection tasks in small microtasks for a text analysis and a model analysis step. We introduced the concept of *Expected Model Elements* (EMEs), a key outcome of a text analysis that represents important model elements derived from a reference document and is used as an input for defect detection in software engineering models (i.e., model analysis). Thus, the CSI process enables distributed and scalable inspection processes (RQ.1). In a feasibility study we evaluate the effects of CSI and P&P in a controlled experiment. We present the study process and report on preliminary results of a large-scale experiment on the effectiveness and efficiency of defect detection for a CSI and

traditional P&P inspection process. The first results indicate that the concept of EMEs helps to improve the defect detection performance for model inspection (RQ.2). Furthermore, we observed comparable results for defect detection effectiveness and advantages for defect detection efficiency (RQ.3).

Future Work. This paper reports on preliminary findings after initial data analysis. An in-depth analysis is planned and required to fully understand the effects of CSI and P&P inspection, e.g., verification of reported and seeded defects, team effects on defect detection performance, and qualification issues.

Acknowledgments.

We would like to thank the participants of the experiment. Thanks also to CAPES and the Brazilian Research Council (CNPq, process number 460627/2014-7) for financial support. We thank Jürgen Musil for his contribution on the overview of tools for software review.

References

1. Anderson P., Reps T., Titelbaum T., Zarins M.: Tool Support for Fine-grained Software Inspection. In: IEEE Software, 20(4), pp.42-50, 2003.
2. Aurum A., Petersson H., Wohlin C.: State-of-the-Art: Software Inspection after 25 years, In: Journal of Software, Testing, Verification and Reliability, 12(3), pp.133-154 (2002).
3. Biffl S., Grünbacher P., Halling M.: A Family of Experiments to Investigate the Effects of Groupware for Software Inspection. Automated SW Eng., 13(3), pp.373-394, 2006.
4. LaToza T.D., van der Hoek A.: Crowdsourcing in Software Engineering: Models, Motivations, and Challenges, IEEE Softw. 33(1):74-80, 2016.
5. Mao K., Capra L., Harman M., Jia Y.: A survey of the use of crowdsourcing in software engineering; In: Journal of Systems and Software, 28p., Available Online: http://dx.doi.org/10.1016/j.jss.2016.09.015, 2016.
6. NASA: Software Formal Inspection Standards, NASA-STD-8739.9, NASA, 2013.
7. Poesio M., Chamberlain J., Kruschwitz U., Robaldo L., Ducceschi L.: Phrase detectives: Utilizing collective intelligence for internet-scale language resource creation. ACM Trans. Interact. Intell. Syst. 3(1), 44p., 2013.
8. Quinn A., Bederson B.: Human Computation: A Survey and Taxonomy of a Growing Field, In Proc. of Human Factors in Computing Systems (CHI), pp.1403-1412, 2011.
9. Rigby P., Cleary B., Painchaud F., Storey M-A., German D.: Contemporary Peer Review in Action: Lessons Learned from Open Source Development. IEEE Software, 29(6), pp.56-61, 2012.
10. Suryn W.: Software Quality Assurance. A Practitioner's Approach, Wiley, 204p. 2014.
11. Travassos G., Shull F., Fredericks M, Basili V.R.: Detecting Defects in Object Oriented Designs: Using Reading Techniques to Increase Software Quality, In: Proceedings of the 14th ACM SIGPLAN OOPSLA Conference, pp 47-56 (1999).
12. Winkler D., F.J. Ekaputra, Biffl S.: "AutomationML Review Support in Multi-Disciplinary Engineering Environments", In: Proceedings of ETFA, pp.1-8, 2016.
13. Wohlin C., Runeson P., Höst M., Ohlsson M., Regnell B., Wessl A.: Experimentation in software engineering, Springer (2012).