# Investigating a Distributed and Scalable Model Review Process

**Dietmar Winkler**
Christian Doppler Laboratory for Security and Quality Improvement in the Production System Lifecycle, Vienna University of Technology, Institute of Information Systems Engineering, Information & Software Engineering,
Vienna, Austria
*dietmar.winkler@tuwien.ac.at*

**Marcos Kalinowski**
Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Department of Informatics,
Rio de Janeiro, Brazil
*kalinowski@inf.puc-rio.br*

**Marta Sabou, Sanja Petrovic, Stefan Biffl**
Vienna University of Technology, Institute of Information Systems Engineering, Information & Software Engineering,
Vienna, Austria
*<firstname>.<lastname>@tuwien.ac.at*

**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. Moreover, from a practical perspective the adapted review process can be used to complement inspection efforts conducted using the traditional inspection process, enhancing the overall defect detection effectiveness. [Conclusions] Although the study shows promising results of the novel process, future investigations should consider larger and more diverse review artifacts and the effect of using limited and different scopes of artifact coverage for individual inspectors.

**Keywords:** Review, Inspection, Models, Model Quality Assurance, Crowdsourcing, Feasibility Study, Controlled Experiment.

## 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 [13] 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 [15]) can support the review process, it is still challenging to review large artifacts, assuring coverage and addressing the most critical system parts. 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 [16].

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) [11], databases, or image analysis [12]. 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 coordination tool support.

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 [17]. 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 process. Second, at a tool support level, we explore the feasibility of implementing key review tasks within the *CrowdFlower*[1] crowdsourcing platform to perform model verification with experts. We evaluate the proposed review process and tool support in a feasibility study, comparing it to traditional P&P based inspections [18].

In this paper we extend our previous publication [18] by providing a more detailed view on the CSI review process, on the feasibility study design and on its results. Furthermore, we also investigate whether the two approaches (CSI and traditional P&P) can complement themselves by analysing the individual defects found by each approach. The study results show the feasibility of the adapted review process and that inspectors using the adapted process achieved comparable results for defect detection effectiveness and better defect detection efficiency. Our findings also indicate that the adapted review process can be used to complement traditional inspection efforts (e.g., for critical software), enhancing the overall defect detection effectiveness by finding additional defects.

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. Next, we describe the controlled experiment in Section 5 and the preliminary results in Section 6. In Section 7 the results are discussed based on general process observations and practical implications to industry practitioners. Finally, Section 8 concludes this paper and summarizes future work.

## 2 Background and Related Work

In this section, we describe related work on Software Reviews and Inspections (Section 2.1) and Crowdsourcing in Software Engineering (Section 2.2).

[1] CrowdFlower: www.crowdflower.com

### 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 inspection process [5]. It consists of five main 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.

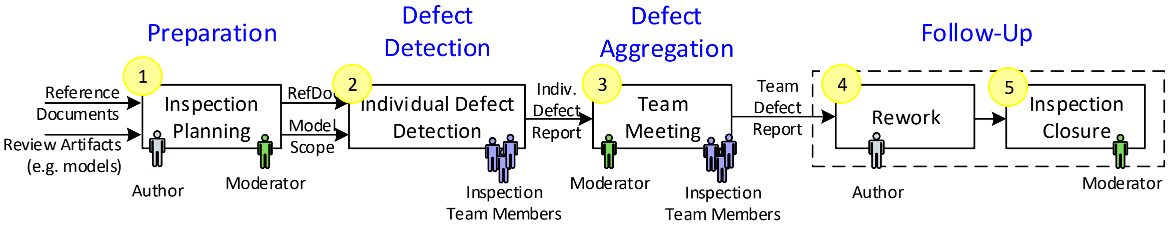

**Figure 1**: Traditional software inspection process [5]

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 [13] 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]. Some tool support has been proposed to support overall inspection process coordination. For instance, the web-based tools presented in [6] and [7] allow reducing inspection meeting effort by supporting a slightly modified inspection process that replaces the face to face meetings with asynchronous discussions. However, those tools do not support scoping during inspection planning for handling large artifacts.

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),

[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

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*[1], have fuelled 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” [8][9].

LaToza [8] 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*[7] 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 [9]. 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 [9] nor in practice. For example, leading software crowdsourcing platforms such as TopCoder[7] do not support software model verification. Our research 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 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. From these expectations 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 present the designed extended inspection process that takes into consideration benefits of crowdsourcing (e.g., microtasking and coordination).

- *RQ.2 What are the effects of the CSI approach with focus on (a) defect detection performance, i.e., defect detection effectiveness and efficiency?* We analysed the adapted inspection approach, i.e., crowdsourcing-based inspection (CSI) in comparison to a traditional P&P inspection process executing a feasibility study (controlled experiment) to investigate defect detection performance.

[6] Mechanical Turk: www.mturk.com

[7] TopCoder: www.topcoder.com

- *RQ.3 Are the CSI approach and traditional P&P inspection processes complimentary with respect to the coverage of defects?* Complementing inspection efforts could be interesting in case of designing critical systems. To answer this research question, we compare the sets of defects found by each of the two approaches.

## 4 CrowdSourcing-based Inspection (CSI) Process

The core idea of the proposed CSI process on how to extend a traditional software inspection process to enable the application of HC&C methods is to split the inspection task into smaller microtasks to allow parallelization of work. As will be detailed hereafter, these microtasks are conducted within a *Text Analysis* phase and a *Model Analysis* phase. The enable splitting the process we introduced the concept of Expected Model Elements (EMEs), a key intermediate outcome of a *Text Analysis* that represents important model elements derived from a reference document, which is used as an input for defect detection in software engineering models during *Model Analysis*. A detailed view on the CSI process follows.

Based on the traditional inspection approach, we focus on the *Preparation* and *Software Inspection* phases (i.e., individual defect detection and team meeting). Fig. 2 presents the adapted CSI process that consists of four phases: (1) *Preparation*; (2) *Text Analysis* to identify Expected Model Elements (EMEs); (3) *Model Analysis* to find defects based on EMEs; and (4) *Defect Analysis and Aggregation*. Note that the *Follow-Up* phase (similar to the traditional inspection process shown in Fig. 1) has been excluded from Fig. 2 because of readability issues.

In the *Preparation* phase, the moderator performs inspection planning and takes, in addition, the CSI management role. The author supports the moderator. Main tasks include (a) scoping of inspection artifacts, (b) preparing the crowdsourcing environment, and (c) uploading reference documents (i.e., a requirements specification) and inspection artifacts (e.g., EER diagrams or UML variants) into the crowdsourcing platform. Therefore, the requirements specification, which is often structured into application scenarios, is split into a set of small entities, e.g., text fragments or sentences. Thus, each sentence represents an input for a microtask for CSI workers and defines the scope for the text analysis.

The *Text Analysis* phase includes the analysis of the reference document with focus on identifying EMEs (Step 2a) and the analysis and aggregation of delivered EMEs (Step 2b). The identification of the EMEs (e.g., entities of the model, their attributes, and relationships between entities) is executed by the CSI workers and the identified EMEs are reported via the crowdsourcing application. The EME analysis and aggregation is performed by the CSI management by removing duplicate EMEs and mapping synonyms. The overall output of the of Text Analysis phase is an agreed and aggregated list of EMEs that represents the input for the next phase.

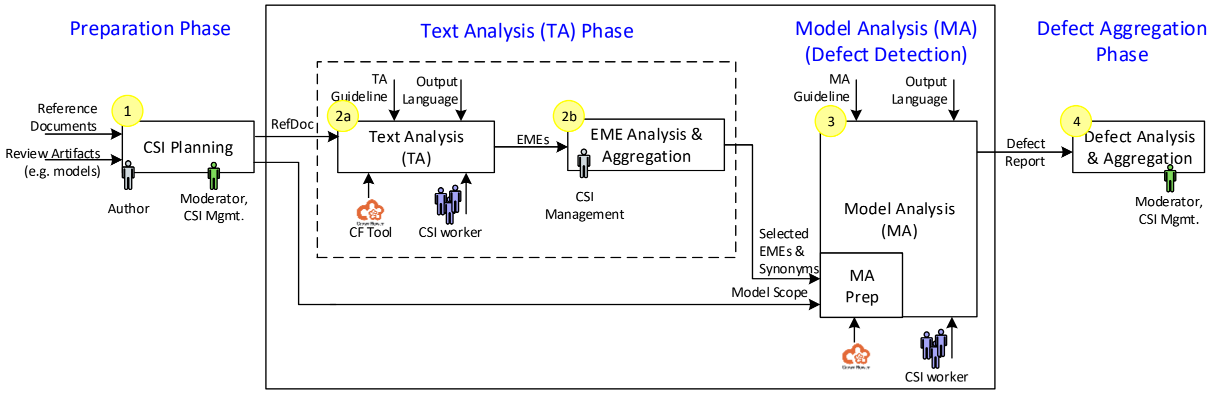

**Figure 2**: Crowdsourced inspection (CSI) process

In the *Model Analysis* phase, the CSI management prepares a selected set of EMEs, derived from manual text analysis and EME aggregation (output of Step 2b) for model inspection. Furthermore, the CSI management prepares the model or a sub-model to be inspected. Sub-models are of specific interest if large models have to be inspected. In context of our study, there was no need for scoping the model because of an acceptable model size. Otherwise, model scoping (or slicing) strategies could be applied [4]. For defect detection (i.e., model analysis), CSI workers receive an EME, e.g., an entity attribute, locate it in the model, and report either that the EME was modelled correctly or report at least one defect. Candidate defects are reported via the crowdsourcing application.

In the *Defect Analysis and Aggregation* phase, the CSI management aggregates reported defects. The subsequent *Follow-up* phase (i.e., rework and inspection closure) is similar to the traditional inspection process (cf.

Fig.1). Note that the author is not included in the crowdsourced defect detection tasks. He receives the aggregated defect detection reports for rework.

For the implementation of the CSI process we used *CrowdFlower* and a compiled a set of sentences to CF jobs for *Text Analysis* and a set of EMEs for the *Model Analysis*. Note that this setting enabled the inspection moderator to (a) balance the work load for the CSI workers and (b) flexibly include additional CSI workers 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 workers represent individual inspectors (or experts) that can be recruited/invited to support the defect detection process, driven by the *CrowdFlower* 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 [19] 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 data screening, assignment of reported defects to reference defects, and evaluation of research questions.

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 (Fig. 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.

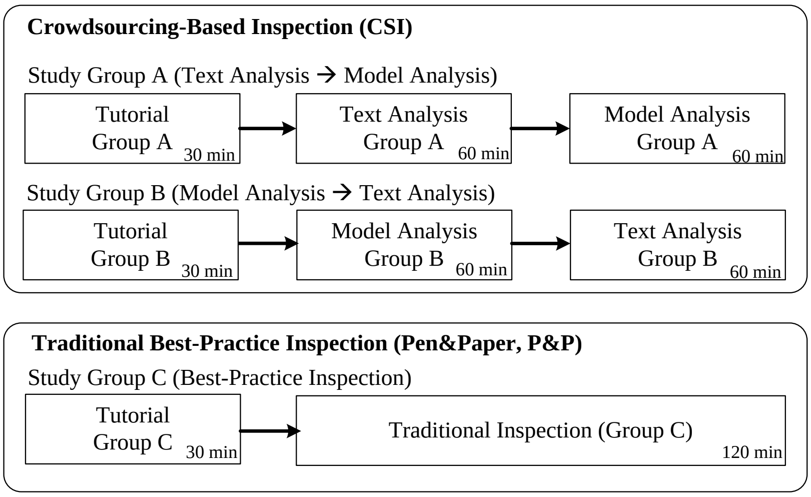
Crowdsourcing-Based Inspection (CSI)
Study Group A (Text Analysis → Model Analysis)
Tutorial Group A 30 min
Text Analysis Group A 60 min
Model Analysis Group A 60 min
Study Group B (Model Analysis → Text Analysis)
Tutorial Group B 30 min
Model Analysis Group B 60 min
Text Analysis Group B 60 min
Traditional Best-Practice Inspection (Pen&Paper, P&P)
Study Group C (Best-Practice Inspection)
Tutorial Group C 30 min
Traditional Inspection (Group C) 120 min

**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 Materials 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.

The seeded defects were introduced by the experiment team (i.e., the authors) based on defects typically introduced during software design activities. This was done by selecting a set of real defects introduced by an independent set of students when building the model based on the reference document. Besides being real, these defects were spread throughout different parts of the model, which was interesting given that we wanted to have CSI workers focusing on finding defects in specific parts.

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. In this paper we report on the findings of the study after the mapping of reported candidate defects and seeded defects. During the analysis some few additional defects might have been reported, those were not considered as reference defects. 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 Results

This section summarizes the findings of the feasibility study with focus on: effort, defect detection effectiveness, defect detection efficiency, and the complementarity of CSI and traditional P&P.

### 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 process 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 CSI and P&P 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). However, some inspectors required more time to complete their P&P task / CSI task. 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. 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 (half the time for defect detection), we believe that the CSI process achieved comparable results for defect detection effectiveness more detailed investigations 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 identifying 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). However, the preliminary observations tend to support our expectations that CSI can support defect detection with crowdsourcing techniques.

### 6.4 Complementarity of CSI and Traditional P&P

Main goal of software inspection is the early and efficient identification of defects. However, an important aspect, in particular concerning the design of critical software, 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.

In this additional analysis we observed that 6 defects were not identified by any P&P inspector and 7 defects were not detected by a CSI inspector. While 4 defects were found by P&P but not CSI, 3 defects were found by CSI but not P&P. Another 3 defects remain unidentified both by P&P and CSI inspectors. Thus, P&P inspectors found 27 defects, CSI inspectors found 26 defects, and both combined found 30 defects.

The preliminary results indicate that a combination of traditional and CSI might be reasonable to cover a large part of the system. Of course, this would require additional resources, which might in turn be justified depending on the criticality of the design of the system under construction.

It is noteworthy also that CSI inspectors could be strategically directed to verify different parts of the reference document in the model aiming at improving the overall model quality assurance (e.g., by providing them EMEs from different scenarios for model analysis). However, this was not directly explored in the context of the conducted study and is subject to further investigation.

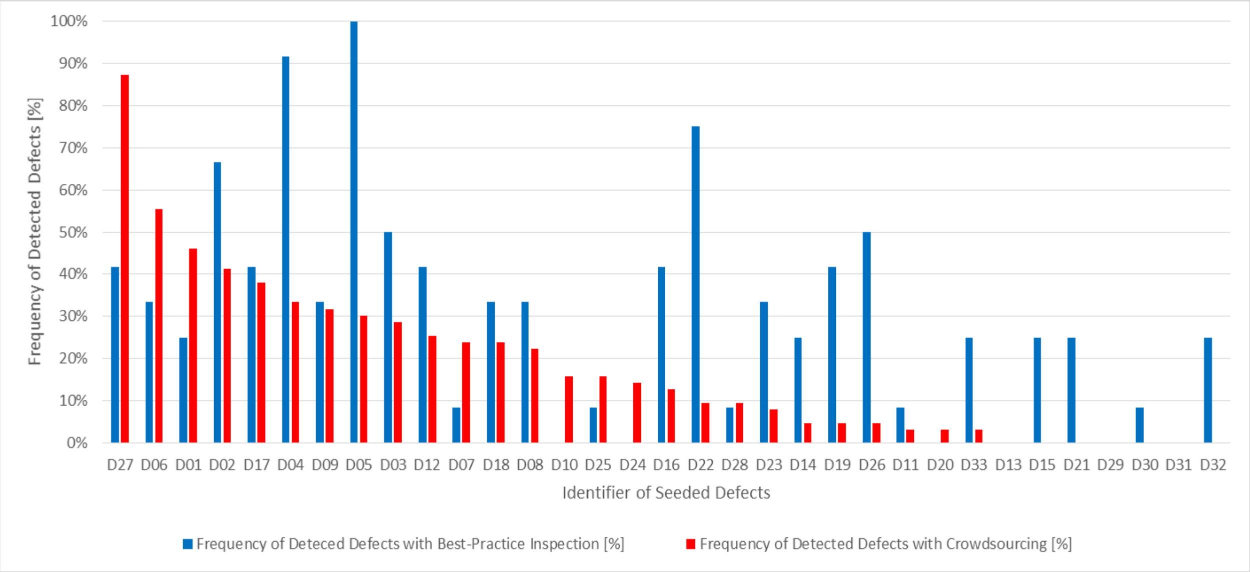

**Figure 4**: Frequency of detected defects [%] by individual inspectors in the empirical study

## 7 Discussion

In this section we discuss general process observations and potential practical implications of the obtained results for industry practitioners.

### 7.1 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 have been discussed with industry and research experts. Table 5 summarizes the main process observations for the needs of software inspection improvement for large and complex software models.

Finally, limited tool support for model inspections hinder efficient P&P inspection while for the CSI process approach appropriate *CrowdSourcing* platforms, such as *CrowdFlower*, can be used to support the inspection process.

**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) CSI workers |
| **Tool Support** | Limited for model reviews | Application of crowdsourcing platforms. |

Regarding *Required Resources*, the application of crowdsourcing platforms such as *CrowdFlower* enables the distribution of derived microtasks among a group of experts and/or CSI workers. Thus, co-located reviews and inspections do not represent a limiting factor.

Concerning the *Review Experience*, smaller tasks also support less and medium experienced inspectors that are guided by the configuration of the microtasks. However, experienced inspectors might be required if a comprehensive view on the overall system is required.

*Defect Detection Guidance* for model inspection in traditional P&P inspection approaches mainly rely on checklists or reading techniques. Available reading techniques are for model inspection are limited and some of them are specific to certain types of models (e.g., OORTs for UML models [15]). The CSI process approach, on the other hand, is driven by EMEs, which represent expected model elements and can be easily adapted for different contexts by changing the types of expected elements. For instance, for the EER diagram of our experiment the EMEs were entities, attributes and relationships, for UML class diagrams the EMEs would be classes, attributes, operations, and relationships. Thus, the CSI approach is more generic and can be easily adapted to be used for inspecting different kinds of models, by simply changing the abstractions to be identified as EMEs.

With respect to *Document Coverage* and *Scalability*, in traditional inspections, where the typical review duration is scheduled for 2 hours of working time, the coverage 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 tasks and the configuration of the inspection process. Thus, the document coverage can be increased by adding more CSI workers or adding additional judgments in case of limited quality.

### 7.2 Practical Implications

From a practical perspective we believe that, especially for critical systems with large and complex models, a combination of traditional and CSI inspection approaches is a reasonable option for improving the overall defect detection effectiveness (by finding additional defects) and helping to cover large parts of the system by involving several/additional CSI workers.

Thus, software organizations could instantiate the CSI process to complement their inspections efforts and improve the overall *document coverage* and defect detection effectiveness. *Required resources* in terms of CSI workers can be experts within the organization, which could even be geographically distributed, assuring *scalability*. While *review experience* is desired, the small CSI tasks also support less and medium experienced inspectors that receive *defect detection guidance* by the configuration of the micro tasks. For *tool support* existing crowdsourcing platforms can be used.

Another noteworthy practical implication is that some basic crowdsourcing tool configuration expertise and potential effort overhead is required for the CSI management role (cf. Fig. 2), which could be performed by the inspection moderator. For instance, during the *Preparation* phase this role is responsible for preparing the crowdsourcing environment, uploading reference document scenarios (split into a set of small sentences) as tasks into the crowdsourcing platform for the *Text Analysis* phase. At the end of *Text Analysis*, this role is responsible for removing duplicate EMEs and mapping synonyms to reach an agreed aggregated list of EMEs that represents the input for the *Model Analysis* phase. Before *Model Analysis*, the CSI management role is responsible for preparing crowdsourcing tasks for the selected set of EMEs and the model to be inspected. Finally, in the *Defect Analysis and Aggregation* phase CSI management aggregates reported defects.

Most of the CSI management tasks can be further supported exploring features that are common to crowdsourcing applications (e.g., for preparing tasks and aggregating results) or even automating new features (e.g., generating crowdsourcing tasks from reference documents and lists of EMEs). Nevertheless, following a scientific approach for developing software technologies, these additional automation efforts should naturally be conducted after further understanding the results and implications of the overall CSI approach through feasibility and observational studies. In this paper we delivered this important first step by providing and discussing results of the feasibility study. Conclusions and an outline of future work follow.

## 8 Conclusions and Future Work

In this paper we provided further details on our proposed CrowdSourcing-based Inspection (CSI) process to support early defect detection of large-scale software engineering artifacts and models. The design of 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 phase. 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, which is used as an input for defect detection in software engineering models during model analysis.

We conducted a controlled experiment involving 75 participants to investigate the feasibility of applying CSI and evaluate its effects compared to P&P inspection process. We presented the study process and reported the experiment results concerning effectiveness and efficiency of defect detection for the CSI and the traditional P&P inspection process. From our overall process observations, we conclude that the designed CSI process enables conducting distributed and scalable inspections (RQ.1). The results also indicate that the concept of EMEs helps to improve the defect detection performance for model inspection, we observed comparable results for defect detection effectiveness and advantages for defect detection efficiency (RQ.2). Finally, for assuring the quality of critical system designs a combination of traditional and CSI inspection approaches represents a reasonable option, improving the overall defect detection effectiveness (by finding additional defects) and helping to cover large parts of the system by involving several/additional CSI workers (RQ.3).

While in this paper we focused on the feasibility and used medium sized artifacts, in future work we plan to investigate how the CSI process can be employed in the context of very large artifacts. For instance, based on selected parts of the reference documents, scoping inspection efforts on smaller parts of a model under inspection. We believe that this could allow exploring the CSI crowdsourcing capabilities to achieve quality assurance of large models beyond the current state of the art inspection possibilities. Another future research direction will focus on finding out how many CSI inspectors (considering inspector capabilities) should process the same task to get a good automated prediction on correct task outcome.

### Acknowledgements

The financial support by the Austrian Federal Ministry for Digital, Business and Enterprise and the National Foundation for Research, Technology and Development is gratefully acknowledged. Thanks also to the Brazilian Research Council (CNPq, process number 460627/2014-7).

### References

[1] P. Anderson, T. Reps, T. Titelbaum, M. Zarins, "Tool Support for Fine-grained Software Inspection," *IEEE Software*, 20(4), pp.42-50, 2003.

[2] A. Aurum, H. Petersson, C. Wohlin, "State-of-the-Art: Software Inspection after 25 years," *Journal of Software, Testing, Verification and Reliability*, 12(3), pp.133-154, 2002.

[3] S. Biffl, P. Grünbacher, M. Halling, "A Family of Experiments to Investigate the Effects of Groupware for Software Inspection," *Automated Software Engineering*, 13(3), pp.373-394, 2006.

[4] L. Briand, D. Falessi, S. Nejati, M. Sabetzadeh and T. Yue, "Traceability and SysML design slices to support safety inspections: A controlled experiment," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(1), 2014.

[5] M.E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, 15(7), pp. 182-211, 1976.

[6] M. Kalinowski, G. Travassos, "A computational framework for supporting software inspections," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, pp. 46-55, 2004.

[7] F. Lanubile, T. Mallardo, "Tool support for distributed inspection," in *Proceedings of COMPSAC*, 2002.

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

[9] K. Mao, L. Capra, M. Harman, Y. Jia, "A survey of the use of crowdsourcing in software engineering," *Journal of Systems and Software*, 28p., Available: http://dx.doi.org/10.1016/j.jss.2016.09.015, 2016.

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

[11] M. Poesio, J. Chamberlain, U. Kruschwitz, L. Robaldo, L. Ducceschi, "Phrase detectives: Utilizing collective intelligence for internet-scale language resource creation," *ACM Trans. Interact. Intell. Syst.*, 3(1), 44p., 2013.

[12] A. Quinn, B. Bederson, "Human Computation: A Survey and Taxonomy of a Growing Field," in *Proc. of Human Factors in Computing Systems (CHI)*, pp.1403-1412, 2011.

[13] P. Rigby, B. Cleary, F. Painchaud, M-A. Storey, D. German, "Contemporary Peer Review in Action: Lessons Learned from Open Source Development," *IEEE Software*, 29(6), pp.56-61, 2012.

[14] W. Suryn, *Software Quality Assurance. A Practitioner's Approach*. Wiley, 2014.

[15] G. Travassos, F. Shull, M. Fredericks, V.R. Basili, "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.

[16] D. Winkler D., F.J. Ekaputra, S. Biffl, "AutomationML Review Support in Multi-Disciplinary Engineering Environments", in *Proceedings of ETFA*, pp.1-8, 2016.

[17] D. Winkler, M. Sabou, S. Petrovic, S. Biffl, M. Kalinowski, G. Carneiro, "Improving Model Inspection with Crowdsourcing," in *Proceedings of the 4th International Workshop on Crowdsourcing in Software Engineering (CSI-SE), ACM/IEEE International Conference on Software Engineering (ICSE)*, Buenos Aires, Argentina, 2017.

[18] D. Winkler, M. Sabou, S. Petrovic, G. Carneiro, M. Kalinowski, S. Biffl, "Investigating Model Quality Assurance with a Distributed and Scalable Review Process," in *Proceedings of the 20th Ibero-American Conference on Software Engineering (CIBSE), Experimental Software Engineering (ESELAW) Track*, Buenos Aires, Argentina, 2017.

[19] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, A. Wessl, *Experimentation in software engineering*. Springer, 2012.