Proceedings of the Eighth AAAI Conference on Human Computation and Crowdsourcing (HCOMP-20)

Verifying Extended Entity Relationship Diagrams with Open Tasks

Marta Sabou,[2] Klemens Käsznar,[2] Markus Zlabinger,[2] Stefan Biffl,[2] Dietmar Winkler[1,2]
[1]Christian Doppler Laboratory CDL-SQI
[2]Institute of Information Systems Engineering, Technical University of Vienna,
Favoritenstrasse 9-11, 1040 Vienna, Austria
{first.last}@tuwien.ac.at

Abstract

The verification of *Extended Entity Relationship* (EER) diagrams and other conceptual models that capture the design of information systems is crucial to ensure reliable systems. To scale up verification processes to larger groups of experts, Human Computation techniques were used focusing primarily on *closed tasks*, which constrain the number and variety of reported defects in favor of easy aggregation of derived judgements. To address this limitation of closed tasks, in this paper, we investigate EER verification (as instance of a broader family of model verification problems) with *open tasks* to extend the range of collected results. We also address the challenge of aggregating results of open tasks by proposing a follow-up HC task for defect validation. We evaluate our approach for HC-based EER Verification with open tasks in a set of experiments conducted with junior developers and show that (1) open tasks allow collecting a variety of insights that go beyond a manually built gold standard while still leading to good performance (F1=60%) and (2) HC-based validation can be reliably used for validating the results of open tasks (F1=84% compared to expert validation).

1 Introduction

As online, distributed work becomes the new normality, understanding how cognitively complex problems can be solved with human computation (HC) approaches as an alternative to on-site activities becomes important. A family of such complex problems that are currently primarily performed on-site is the *verification of conceptual models* which are key both in designing information systems (e.g., various system diagrams) and in representing relevant knowledge (e.g., domain models such as ontologies). Ensuring the quality of these models is of major importance as it has a direct effect on the enabled information system.

We hereby focus on the concrete problem of *Extended Entity Relationship (EER) diagram verification*. An EER (Fig. 2b) is a high-level model that describes *entities*, *attributes* and *relations* between entities for database design (Thalheim 2009). The correctness of an *EER model* is success-critical for information systems because defects can have a major impact on derived models, database structures, queries and code. Defects detected late in the (software) engineering project can result in high rework effort, cost and project delays (Boehm and Basili 2005). Therefore, the verification of (EER) models is key to mitigate the risk of high rework. Yet, current EER verification processes performed *on-site* by expert groups suffer from limited expert availability, low scalability and lack of tool support for advanced process control (Winkler et al. 2017). HC techniques could address these issues by scaling up the process through distribution to a larger base of junior software developers working *off-site* through online, distributed work.

A number of works have investigated the use of HC methods for the verification of EER diagrams (Sabou et al. 2018b) and more broadly of conceptual domain models such as ontologies (Acosta et al. 2016; Mortensen et al. 2016). These works typically adopt a HC task design based on *closed tasks*, which constrain workers to choose between a limited number of categories related to the quality of model elements. While this approach facilitates the easy aggregation of results, by design, it limits the ability to capture insights from workers that go beyond what task owners expect to elicit. As an alternative, *open tasks*, in which contributors can provide their observations in (nearly) free text, are better suited to collect a broader set of answers and subtle insights (Eickhoff and de Vries 2013), yet the aggregation of their results is challenging. In this paper, we investigate a HC design based on open tasks and aim to answer the following research questions:

- **RQ1. Model analysis.** To what extent can open tasks capture expert insight while eliciting high quality contributions for defect detection and reporting?
- **RQ2. Defect report validation.** To what extent can HC tasks validate and support the aggregation of defect reports obtained with open tasks?

To answer these research questions, we propose an open task based HC approach for EER model verification and make use of it in an expert-sourcing setting to collect free-text defects about an EER model. To foster the interpretation of the collected output as well as follow-up aggregation steps, an expert (1) evaluates the quality of proposed defects and (2) aligns correct defects to known *True Defects* in the

model. Based on the expert evaluation we derive insights related to RQ1 in terms of the correctness of the proposed defects (both individually and after aggregation) and the level to which these go beyond a known set of defects. Secondly, to answer RQ2, we introduce a further HC task in which expert-sourcing is used to evaluate the quality of the defects collected with open tasks and then compare this quality to the validation results of the expert. Accordingly, the paper makes the following contributions:

- We report on an *approach to collect defect reports using open tasks* and find that such an approach leads to high quality results while eliciting also defects beyond those known by the task owners;
- We present an *approach to evaluate the defect reports* obtained with open tasks through a follow-up HC task and show that this leads to comparable performance (precision/recall) with those obtained by an expert evaluator.

After discussing related work (Sect. 2), we provide details of the EER diagram verification problem (Sect. 3) and our proposed approach to collect and verify defects (Sect. 4). Sections 5 and 6 report on the experiment setup and discuss the obtained results. Sect. 7 concludes the paper.

2 Related Work

The use of crowdsourcing techniques is widely spread to support tasks in all phases of the *Software Engineering* Life-Cycle (Mao et al. 2017). Yet, EER diagram verification has received little attention beyond our earlier work where it was addressed by using closed tasks (Sabou et al. 2018b).

The situation is similar in *Knowledge Engineering* where an increasing number of approaches focus on solving a *similar problem* of verifying *domain* models (Sabou et al. 2018a). Domain (as opposed to system) models such as ontologies, taxonomies or knowledge graphs need to be verified for their correctness, as the following works exemplify.

With the goal of evaluating the quality of triples from *Linked Data Knowledge Graphs*, namely *DBpedia* (Auer et al. 2007), Acosta et al. (2016) enlist both experts and layman crowds. A two-stage *Find-Verify* workflow is used: first, workers select one of three possible quality issues that could apply for a given triple (i.e., value, link, datatype); second, they solve tasks in jobs focused to individual quality issues and provide a binary judgement of the triples' correctness with respect to that issue. The goal of Mortensen et al. (2015; 2016) is verifying the correctness of subsumption relations in large medical ontologies.The verification tasks contain the relation to be evaluated described in natural language and the definition of concepts connected by that relation. Workers make a binary choice on the correctness of the relation and provide an explanation. Wohlgenannt et al. (2016) propose a plugin for a popular ontology development tool that allows ontology engineers to easily create HC tasks for verifying various aspects of the ontology such as the domain relevance of terms or relation correctness. All generated tasks are closed tasks that collect binary decisions.

To conclude, both in *Software and Knowledge Engineering*, HC approaches that verify some aspect of a conceptual models (EERs, ontologies) use closed tasks to collect (binary) decisions about the correctness of a model element.

While closed task types are widespread in crowdsourcing projects (Jain et al. 2017) thanks to the ease of aggregating their results, *open task* designs that provide higher worker autonomy for performing the task have been shown to (i) lead to *higher intrinsic worker motivation* as workers can satisfy needs of self-expression and experience feelings of free choice and commitment (Moussawi and Koufaris 2013) and (ii) *foster creativity while deterring cheaters* (Eickhoff and de Vries 2013). Therefore, based on the works above, in this paper, we go beyond the current practice of using closed tasks for solving model verification problems and investigate open tasks instead, with a focus on the concrete problem of EER model verification.

3 EER Diagram Verification

EER diagram verification focuses on checking how an *EER diagram* confirms to a *reference document*, e.g., a textual system specification, which describes a target software system. As a result of this verification process, a number of *defects* are identified pinpointing modeling errors in the diagram. For example, for our experiments we verify the *EER diagram* of a restaurant ordering system in terms of a corresponding system specification (see snapshots in Fig. 2a,b). Formally, the EER diagram verification refers to checking whether an EER model M covers completely and correctly a frame of reference (FR), such as a specification document. It is a function γ that given M and FR returns a list D of defects in M: $\gamma(M, FR) \rightarrow D$.

Given its importance, the EER verification task is performed in practice by following a number of established methods and processes. In particular, *Software Reviews* and *Software Inspections* are well-established approaches for detecting defects in (Software Engineering) artifacts efficiently and effectively (Aurum, Petersson, and Wohlin 2002). In the context of Software Inspection, important process steps consist of an individual defect detection process step, executed by a team of experts and a team meeting to validate and aggregate individual candidate defect lists with the goal to converge to an agreed set of (true) defects (Laitenberger and DeBaud 2000). *Reading techniques* support individual inspectors in their defect detection tasks by providing guidelines for defect detection, e.g., checklist-based, perspective-based, usage-based reading techniques (Kollanus and Koskinen 2009). Several authors report the use of inspections and reading techniques for EER-diagram verification (Hungerford, Hevner, and Collins 2004; Wohlin and Aurum 2004).

Although traditional software inspection has many benefits for defect detection, it also faces limitations such as: *(i) lack of experts*, which are expensive and scarce resources, to conduct defect detection and validation; *(ii) limited scalability of inspection processes* typically scheduled for two hours thus constraining the scope of the inspection artifacts; *(iii) challenging inspection control and coordination*, due to lack of suitable tool support which leads to inspection processes being planned and executed without *in-process* feedback, e.g., for clarifications or coordination (Winkler et al.

2017). HC techniques could help to overcome these limitations by involving groups of junior software developers working off-site, scaling up the process by distributing the tasks over a larger expert base and by providing tooling for in-process control and improved coordination.

4 HC Approach to EER Verification

Based on traditional software inspection (Aurum, Petersson, and Wohlin 2002), we propose the *Crowdsourced Software Inspection* (CSI) process to the problem of EER diagram verification (Fig. 1) including four steps: (1) *CSI Planning*, where a moderator plans an inspection task and selects the corresponding artifacts for defect detection (M,FR); (2) *Model Analysis* for individual defect detection to collect candidate defects reported by individual workers (Sect. 4.1); (3) *Defect Validation*, where a group of workers validates reported candidate defects (Sect. 4.2); and (4) *Follow Up* where the moderator uses the inspection result for initiating defect repair and decision making, typical activities of software inspections (Aurum, Petersson, and Wohlin 2002).

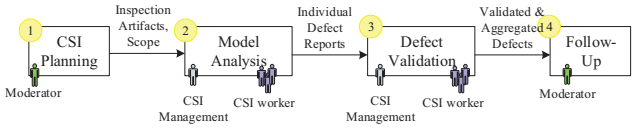


Figure 1: Crowdsourced Software Inspection (CSI) Process.

4.1 Model Analysis Tasks

In order to break up the EER verification problem into smaller tasks that are amenable to HC, we see an EER model as a collection of *model elements* (me) and focus the inspection of the model M on the inspection of its elements. Formally, $M = M_E \cup M_R \cup M_{EA} \cup M_{RA} \cup M_{RM}$, where:

- M_E is the set of all **entities**, e.g., *customer, order, invoice*;
- M_R is the set of **relations** declared between entities, e.g., *(customer, orders, order)*;
- M_{EA} is the set of all **entity attributes**, e.g., *name* in *customer.name*;
- M_{RA} is the set of all **relation attributes**, e.g., *date* in *(customer, orders, order).date*;
- M_{RM} is the set of all **relation multiplicities**, e.g., *1, 0..n* in *customer(1), orders, order(0..n))*.

Task Input. According to this approach of splitting the EER diagram verification task, we implement a *Model Analysis Task*. We prepare individual HC tasks that focus on a combination of a model element me and a corresponding snippet from the reference document FR. To that end, the identification of mes was performed manually by extracting model elements that are mentioned in the specification document. Spurious mes from M were not part of the input thus curtailing the identification of $Spurious$ type defects. Subsequently, mes are mapped to suitable evidence. In our case, we assign to each me a paragraph (corresponding to a scenario sc) in the specification document, where the me is described. Therefore, the input to the task are (me, sc) pairs and the model M. Please refer to Fig. 3 (explained in Sect. 6) for an overview of the data processing workflow of the data items introduced in this section.

Task Design & Output. A HC-task focuses on an me that is judged, showing the model M as well as the scenario in which the me appears (Fig. 2a-c). Workers are first asked to identify if a specific me is modeled in M and whether there are defects related to that me. In this case, the defect(s) should be specified in a textual format using a proposed structure that includes the following fields:

- *expected modeling*, i.e., the expected correct me, e.g. $skiingTicket.timeStamp$ is expected to be a key;
- *defect description* in the current modeling of the me, e.g., $skiingTicket.timeStamp$ is not a valid key;
- *defect severity*, which can be critical, important or minor.

As output of this task, for each (me, sc) pair, the analysis workers ($w_a \subset W_a$) either identify that (i) no defect has been found in the model, leading to an *individual no-defect report* $INDR(me, sc, w_a)$ or (ii) one (or more) defect candidate(s) have been found, leading to an *individual defect report* $IDR(me, sc, w_a, D_{text})$, where D_{text} should comply to the defect description template (see Fig. 2d).

4.2 Defect Validation Tasks

While open tasks foster collecting unbounded worker insight, the aggregation of the collected textual defect reports (IDRs) cannot be performed automatically. Therefore, an additional validation stage is introduced, in which collected IDRs are (i) *validated* for their correctness (i.e., that they indeed represent defects) and (ii) *aligned* to a corresponding *True Defect* (TD) from a known gold standard for that me in order to enable aggregation of the collected judgements. We experiment with two approaches for defect report validation:

- *expert-based* approach (Fig. 3/step 2), in which a domain expert manually inspects IDRs and assigns them a correctness value and possibly a corresponding TD;
- *crowd-based* approach (Fig. 3/step 7), in which workers (W_v) validate and assign an IDR to a TD. In the following, we discuss the design of the *Defect Validation* task.

Task Input are the IDRs collected during *Model Analysis*. Additionally, as preparation for the TD assignment task, for each IDR, we select a TD that is declared at that me. Alternatively, if there is no TD declared at the me, we identify a TD declared at an mes in immediate vicinity (e.g., entities and their attributes; relations, their attributes and multiplicities) based on observations that several IDRs declare defects at neighbouring entities (see Sect. 6.1). Therefore, one IDR may be sent for validation with several TDs. In these cases, as many tasks are created as corresponding TDs found. For example, there are a number of IDRs declared for the relation $(order, orderFoodItem, FoodItem)$, yet no TD refers to this me in the *True Defect Catalogue*. At the same time, there are three TDs declared in the vicinity of the relation (see Table 1) referring to its *multiplicity* (D23), to

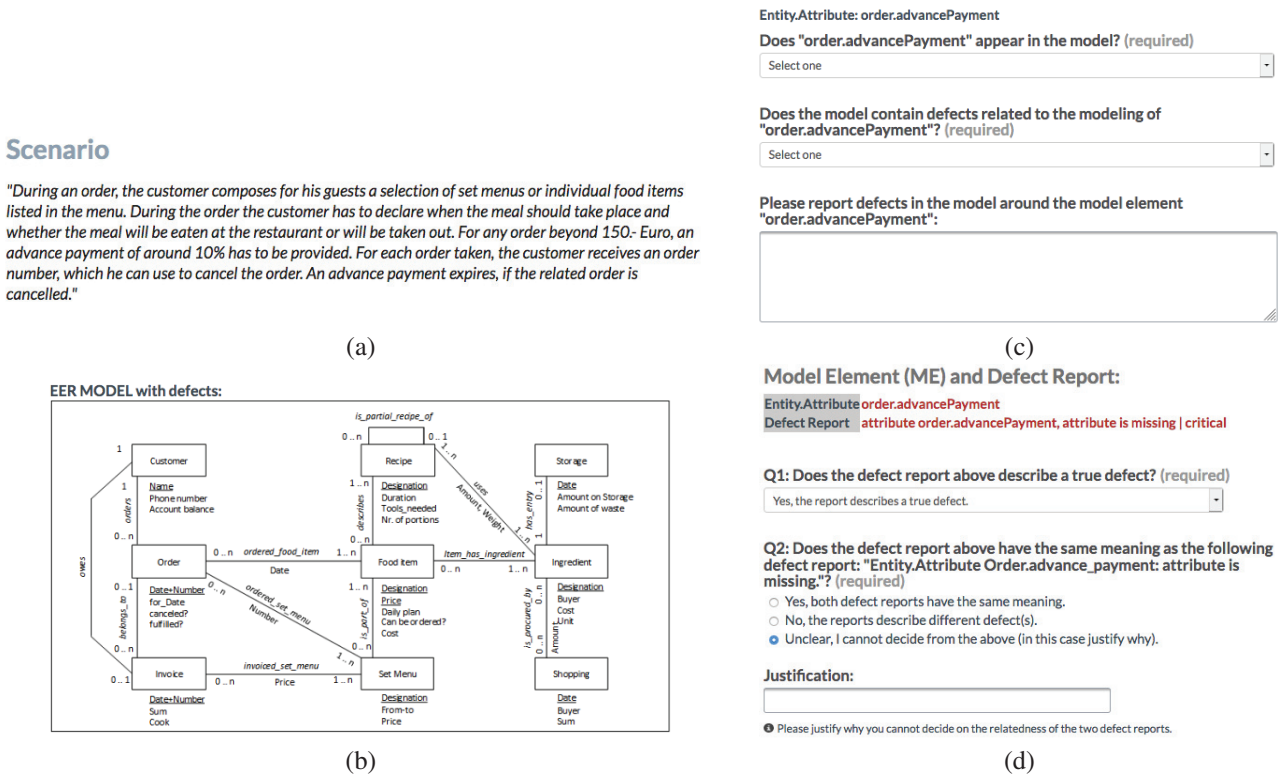


Figure 2: (a) An evidence scenario; (b) the EER model with defects; (c) *Model Analysis* focus entity and questions; (d) *Model Validation* focus entity, defect report at this element to be validated and questions for guiding the validation process.

the missing relation attribute $number$ (D25) and the superfluous $date$ attribute (D24). Therefore, each IDR declared at the relation level might refer to one of the TDs D23, D24 or D25. Accordingly, each such IDR is verified as part of 3 different tasks, one for each TD. IDRs for which no TD is declared at the me or neighbouring mes are not sent for crowd-validation, as they are deemed potentially incorrect.

Task Design & Output. We design a task (see Fig. 2a,b,d), which depicts the evidence scenario, the EER diagram, the model element that is validated and a D_{text} that was collected as part of a model analysis task. Through question Q1 (Fig. 2d) workers assess the truth value of the IDR. If the IDR is considered correct, a follow up question Q2 is displayed which depicts a potential TD description that the IDR might capture, allowing workers to specify whether the IDR refers to the TD or not. Alternatively, if a decision cannot be taken, a justification is given. By interpreting the responses to these questions, the output of this task is an *individual defect validation* $IDV(IDR_{me}, w_v, C, [TD_{id}])$ of an IDR reported at a me by a validation worker (w_v), with labels assigned that indicate the correctness of the IDR (C), and, if a suitable TD was found, the identifier of the TD (TD_{id}).

5 Experimental Setup

To assess the quality of judgements collected with open tasks (RQ1) and to investigate the feasibility of crowd-based validation (RQ2), we set up a large-scale controlled experiment in an academic environment following guidelines proposed by (Wohlin et al. 2012). We collected data in three experiment runs at an academic course on *Software Quality Assurance*: a first Model Analysis experiment involved 54 participants and provided the input for two Defect Validation experiments with 54 and 75 participants respectively.

Experiment Process All experiment runs followed a similar process: (a) a *preparation* phase, focuses on setting up the controlled experiments including material preparation and tooling, tested in pilot runs to ensure the correctness and the feasibility of the experiment; (b) *execution and data collection* phases focus on *model analysis* (first experiment) or *defect validation* (second and third experiment); and (c) *data analysis and interpretation* include evaluation steps for investigating RQ1 and RQ2 (see Fig. 3). Participants received 30 minutes training on *model analysis* and *defect validation* tasks and the *CrowdFlower* application, which provided the corresponding HC tasks. For training, we used a similar setup but focused on a different example from a known application domain (i.e., processes in a parking garage). Both *model analysis* and *defect validation* tasks were scheduled for 60 minutes working time.

Experiment Participants All experiments were executed in an academic environment at TU Wien, involving a total number of 183 participants. The authors are aware of issues with student experiments (Runeson 2003). However,

Table 1: Example defects from the True Defect Catalogue.

ID	Model Element (me)	Defect Description
D23	Relationship Order-food_item	multiplicity should be fooditem(0..n) not fooditem(1..n))
D24	Relationship.attribute Order-food_item.date	attribute is superfluous and misleading
D25	Relationship.attribute Order-food_item.number	attribute is missing

we applied an experience questionnaire prior to all experiment runs to capture background knowledge with focus on (a) industrial project experience in years and (b) software development experience based on a Likert-scale from 0 (no experience) to 5 (professional experience). Analysing *industrial project experience*, 7 participants (4%) did not report any project experience, 134 participants (73%) reported 1-5 years of industrial project experience, 31 participants (17%) reported 5-10 years of industrial project experience, and 11 participants (6%) were involved in industry projects since more than 10 years. With focus on software development experience, 16 participants (9%) assessed themselves as less experienced, 122 participants (66%) reported a medium experience level, and 45 participants (25%) assessed themselves as highly experienced. These analysis results show that most of the participants had several years of experience based on individual (part-time) working practices and, therefore, can be considered as junior software developers.

Experiment Material We applied a study package that has been intensively reviewed by experts and used in several experiment runs containing the following material.

Questionnaires. To capture background knowledge of participants, we applied an *experience questionnaire* prior to the study. After completing the *model analysis* and *defect validation* tasks, we applied feedback questionnaires to capture subjective experiences from participants.

Textual requirements. Study material was a textual reference document, i.e., a system requirements specification including 3 pages in English language, consisting of 7 scenarios and approximately 120 mes, where scenarios were used as frame of reference in both task types.

EER model. We used a medium-scale EER Diagram consisting of 9 entities, 13 relationships and 32 attributes.

True Defect Catalogue is a manually collected set of *True Defects (TDs)* in the *EER model* (see examples in Table 1). Note that the *textual requirements* are considered to be correct. These defects were introduced by the experiment team (including some of the authors) based on pilot trials and complemented with typical defects that occurred in context of such applications during a software engineering process. For each TD we specify: (1) a unique identifier id, e.g., D11; (2) the me it refers to, e.g., $customer.name$; (3) a defect type, which can be one of $Missing$ (the me is not modeled), $Wrong\_Key$ (an entity attribute is not unique and should not be a key), $Superfluous$ (an entity is modeled although it is not relevant), $Wrong\_RelM$ (wrong relation multiplicity declared) or $Wrong$ (a general defect); and (4) a description, e.g., *"Entity.Attribute Customer.Name: Customer name is not a valid key, not unique"*. The *EER model* contains 35 seeded defects, of types $Missing$ (13), $Wrong\_Key$ (5), $Wrong\_RelM$ (7), $Wrong$ (1) and $Superfluous$ (9). In the experiments, these defects represent the gold standard and serve for assessing the quality of the collected data.

Data Collection, Analysis and Variables Experiment data has been organized and collected via the *CrowdFlower* crowdsourcing platform. *Independent variables* are the set of defects seeded in the *EER model*, defect types (i.e., such as missing, wrong, and superfluous information), tool configuration (i.e., configuration of tasks and jobs in *Crowdflower* and the assignment to participants) and the study treatment (i.e., the application of the *HC-based EER Verification approach*). *Dependent variables* are the effort for task execution (not evaluated in this paper), the number of reported and validated defects and true defects. We compute the precision, recall and F-measure metrics.

6 Experimental Results

Fig. 3 depicts the main processing steps taken to interpret the experimental data. Steps 1 to 6 focus on the evaluation of the *model analysis* task as per RQ1 (Sect. 6.1), while steps 7 to 10 cover the evaluation of *defect report validation* (RQ2).

6.1 Model Analysis Results

1. Model Analysis was performed for 120 mes. The output consisted of 1,549 judgements that could be classified either as IDRs (849) or INDRs (700), which were processed separately in Step2 and Step3 as discussed next. Several IDRs provided more than a defect, so these were split prior to the next step, resulting in 963 atomic IDRs.

2. Expert-Based Defect Validation during which an expert evaluated the 963 atomic IDRs. Firstly, he assigned a correctness value (C) of *True Positive* (TP) to 408 IDRs describing a true defect and *False Positive* (FP) to 516 IDRs that did not identify a true defect. 39 IDRs were correct defects that were not identified in the TD catalog (considered as TPs for our analysis). These IDRs are particularly interesting as they show that open tasks allowed to elicit results that went *beyond* an expert-group created gold standard.

Secondly, the expert assigned to 408 TPs a corresponding True Defect id from the TD catalogue. In line with software quality assurance goals of maximizing the number of defects detected, the expert took a lenient approach when assigning TDs to IDRs: he assigned a TD to any IDR that correctly identified that TD even if parts of the IDR might have been incorrect (e.g., the explanation was incorrect or the TD did not strictly refer to the focus me).

The TD assignment revealed a recurring phenomenon of workers tending to *declare defects not strictly at the focus* me, but also at neighbouring mes and randomly (in 264 cases). For example, when asked to judge an me of type *entity*, workers reported defects related to the entity's attributes (which in our conceptualisation are different mes): during the inspection of the $FoodItem$ entity, defects were

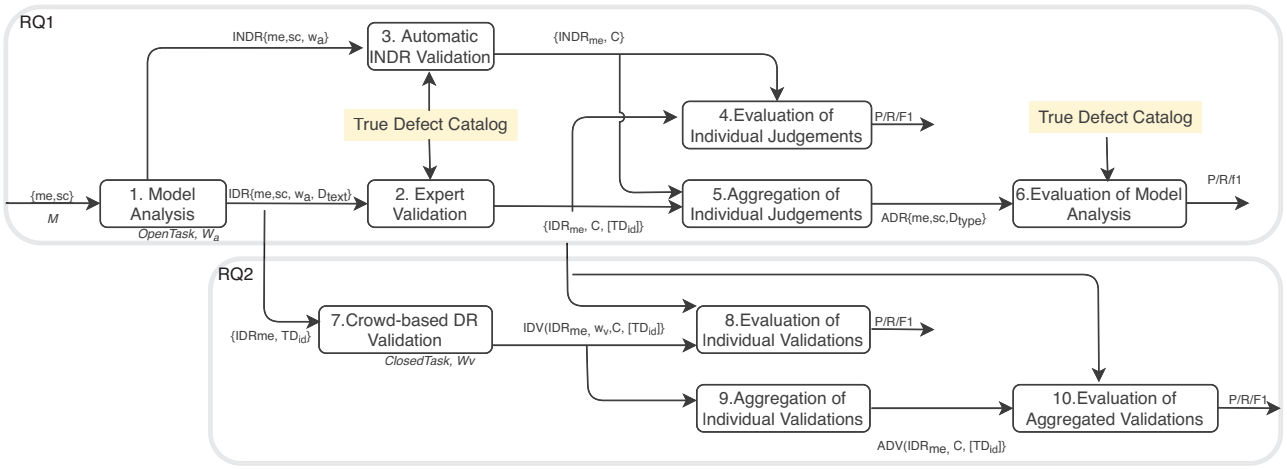


Figure 3: Data collection and processing steps for answering the research questions of the paper.

declared related to the entity attributes $foodItem.price$ and $foodItem.designation$. The same phenomenon emerges for relations and relation-attributes. IDRs referring to neighbouring mes were also considered a correct TP.

Inter-expert agreement of 92% was reached on a number of 25 randomly selected judgements evaluated by a second expert. Due to this high agreement, most of the validation was performed by a single expert.

3. Automatic INDR Evaluation compares those judgements that did not highlight a defect against the TD catalogue and assigns a correctness value (C) as follows: INDRs declared for a me for which no TD is available, are considered *True Negative* (TN), i.e., the worker correctly identified no modeling error; otherwise, if a TD is known at the me, then the INDR is a *False Negative* (FN), i.e., the worker failed to identify a known defect.

From 700 INDRs, 622 (89%) were classified as correct TNs and 78 (11%) were assigned to me's where there is a known defect (FN). An analysis of the "ignored defects" (FN) showed that they affected 20 of the 120 input mes. In some cases, such FNs are due to low-worker performance. However, in a number of cases our analysis revealed that these FNs occur when there are ambiguities in the specification or the labels of the me's. A good example is $storage.date$, at which a $Wrong\_Key$ defect is expected. Although 8 workers have identified this defect, 12 workers did not. Some of these explained their decision due to lack of clarify of the specification, e.g., *"not knowing the exact purpose of the storage entity, I cannot tell whether the primary key on the date field is correct or not"*. This shows another benefit of open tasks: enabling workers to pinpoint potential issues with the quality of the input data.

4. Evaluation of Individual Judgements is performed based on judgement correctness values (TP, FP, TN, FN) assigned by the expert (step 2) and through automatic INDR analysis (step 3). We compute a **P**recision (TP/(TP+FP)) of 46%, showing that about half of the identified defects are correct (see Table 2). **R**ecall (TP/(TP+FN)) is 85%,

	P	**R**	**F1**	**Acc**
IDRs (step 4)	46%	85%	60%	64%
ADRs (step 6)	44%	87%	59%	66%
IDVs C (step 8)	82%	66%	73%	65%
IDVs TD (step 8)	88%	38%	53%	53%
ADVs C (step 10)	87%	81%	84%	77%
ADVs TD (step 10)	97%	53%	69%	63%

Table 2: Overview of performance evaluation metrics.

thus workers identified a high number of the defects that could be found. **F1** measure (2PR/*P+R)) is 60%. **A**ccuracy ((TP+TN)/(TP+TN+FP+FN)) is 64%, which shows that the workers provide a correct judgement in more than half of the judged IDRs. The analysis of worker performance in terms of F1 (Fig. 5a) shows a tendency to perform with F1 above 0.5, however, there are several low performers as well.

We also investigate the extent to which 35 TDs of the gold standard were identified, by displaying how often each TD appears in the set of IDRs (Fig. 4, blue bars). The TDs are also grouped by their type. We find a *high coverage* of the TD Catalogue, with 31 TDs (out of 35) being identified as follows. All defects of type $Wrong\_Key$ and $Wrong\_RelM$ have been identified with a high frequency (each defect identified at least 10 times), indicating that these are the defect types that are easier to find. For $Missing$ defects, 10 (out of 13) defects were identified.

Interestingly, 8 out of the 9 $Spurious$ defects were also identified (although with lower frequencies of < 10), even if the HC task was conceived such that workers were not explicitly asked to consider the spurious elements in M. Additionally, 39 TP defects were found that indicate defects beyond those in the GS. These two defect categories (TP, $Superfluous$) indicate that open tasks lead to results beyond the gold standard. This is particularly useful in software inspection, where identifying as many defects as possible is important.

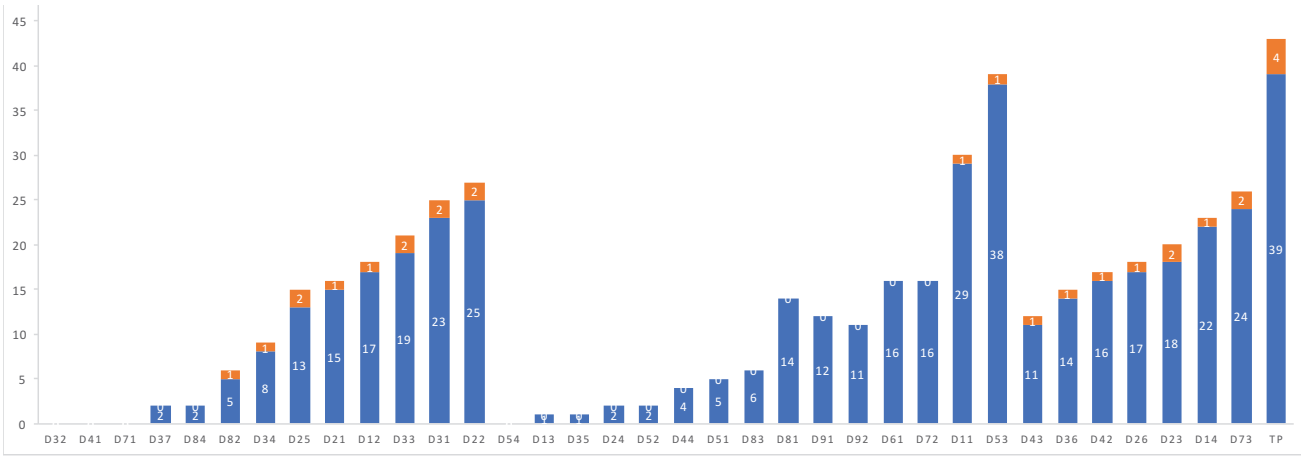


Figure 4: Frequency of identified IDRs (blue/lower bars) and ADRs (orange/upper bars) per defect type $Missing$ (D32-D22), $Superfluous$ (D54-D81), $Wrong$ (D91), $Wrong\_Key$ (D92-D53), $Wrong\_RelM$ (D43-D73) and $TruePositives$ (TP).

5. Aggregation of Individual Judgements is performed to identify *aggregated defect reports* (ADR). Through this, we aim to assign to each (me, sc) pair a final defect type based on majority agreement across workers, that is $ADR(me, sc, D_{type})$. A D_{type} can be (i) a TD id; (ii) NoDefect (if no defect was reported by the majority of workers) or (iii) Undecided (in case of a tie). Currently, we apply a majority voting approach, taking the label with the highest number of votes. For the 120 mes, we converge to 27 correct defects reported, 23 overlapping with the TD catalogue and 4 TPs "emerging" from the crowd. Also, for 52 eme a NoDefect label could be converged to, which in 48 cases was correct (TN) and only in 4 cases incorrect (FN), thus showing a very good performance in identifying what is not a defect. Furthermore, the 4 FNs result from the 39 FNs we identified at step 3 and all refer actually to mes where the specification was ambiguous and could be an important information to consider in order to revise not only the model but also the specification. For 34 mes, incorrect defects were reported (FP). For 7 mes, no decision could be reached.

6. Evaluation of Model Analysis With the final set of aggregated defect reports which constitutes the crowd-response to the model analysis task, we can evaluate the quality of the model analysis by contrasting the obtained results with the True Defect Catalogue. We reach P=44.3%; R=87.1%; Acc=66.4%; F1=58.7%.

In terms of the coverage of the TDs, Fig. 4 (orange bars) shows which defects appear in the aggregated set (also in comparison with the IDR set discussed at Step4). We identified 17 out of 35 defects in the TD catalogue. Some defect types were easier to identify than others. Namely, we identified: all $Wrong\_RelM$ defects; 8 (out of 13) $Missing$ defects; 2 (out of 5) $Wrong\_Key$ defects. The only $Wrong$ defect was not identified nor were the 9 $Spurious$ defects. While 11 defects were only found once, 6 defects were identified on two mes by virtue of reporting defects at neighbouring mes. Additionally, 4 new defects (TPs) emerged in the final set as possible extensions to the TD Catalogue.

6.2 Defect Validation Results

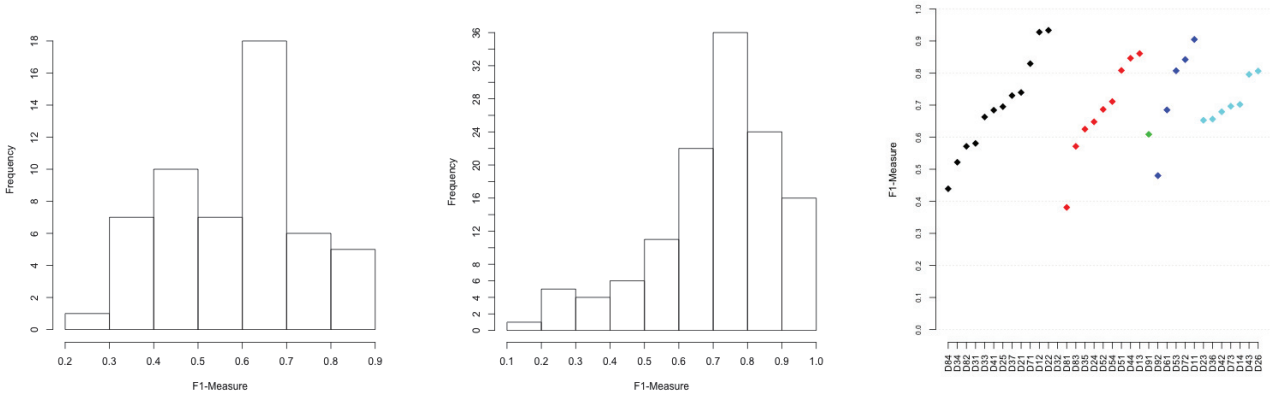
7. Crowd-Based Defect Report Validation focused on a subset of 310 individual defect reports (IDR). These IDRs were matched to the ids of TDs (declared at the me and neighbouring mes) thus resulting in 438 tasks. 125 workers participated and derived 2565 individual defect validation (IDV) judgements, from which 781 IDVs also assigned a potential corresponding TD to 233 of the validated IDRs.

8. Evaluation of Individual Validations aims to find out how individual workers perform, when assessing the IDRs. To that end, we compare their judgements with the expert based evaluation of IDRs in two separate analysis.

First, we investigate the performance of *identifying the correctness value* of an IDR (i.e., agreement between workers and expert as of whether an IDR is a correct defect or not) and reach P = 82%, R = 66%, F1 = 73%, Acc= 65%, for the overall judgement set. Per worker performance as F1 in Fig. 5b shows that workers perform better at defect verification rather than defect identification tasks (shown in Fig. 5a). In terms of factors that might have an influence on worker performance, we found that: (i) for the judged me types, defects related to entity attributes lead to highest performance (ii) for the judged defect type, performance is highest for $Wrong\_Key$ type defects while the performance is comparable for defects of type for $Missing$ and $Wrong\_RelM$. Fig. 5c shows variations of F1 across individual defects, indicating that, in an industrial setting, certain defect type reports should be judged by a senior expert, while other defect types might be solved in a distributed setting with junior developers (in our case $Wrong\_Key$ and $Wrong\_RelM$).

Second, we assess the performance in *assigning the correct TD ids* and obtain P=88%, R=38%, F1=53% and Acc=53%, thus showing that the assignment of the correct defect id can be achieved with a high precision but at the expense of recall.

9. Aggregation of Individual Defect Validations leads to an aggregated defect report validation (ADV), assigning to



(a) (b) (c)

Figure 5: Worker F1 performance for (a) model analysis; (b) defect validation and (c) the validation of concrete TDs grouped per type: black=$Missing$; red=$Superfluous$; green=$Wrong$; blue=$Wrong\_Key$; cyan=$Wrong\_RelM$.

each IDR a correctness value (whether it is a defect or not) and potentially a defect type. Currently, we apply a majority voting approach where each IDR is assigned to the most frequent opinion. From the 438 tasks, we reached a decision for 365 tasks while 73 tasks remain undecided.

10. Evaluation of Aggregated Validations in comparison with the expert validation was split on (i) the task of assigning the correctness values, leading to P=87%, R= 81%, F1=84%, Acc=77%, showing a high quality of the aggregated results with respect to the expert; (ii) the task of also assigning equivalent TDs, which lead to P=96.7%, R= 53%, F1=68.7%, Acc=63%. These results indicate that it would be feasible to replace expert based validation by the follow-up HC task, especially for deciding the IRD's correctness.

7 Conclusion and Future Work

In this paper, we focused on addressing the problem of EER diagram verification with open task-based HC. **RQ1** referred to the model analysis performance obtainable with such task types. Through quantitative and qualitative analysis of data gathered from a controlled experiment we conclude that open tasks support collecting a broad range of insights from participants: (i) we collected *new defects* which appeared both at individual judgement (IDR) level and persisted after aggregation (ADR level); (ii) workers identified $Superfluous$ defects even if they were not explicitly asked to verify these model elements - however, this phenomena is observed only at the individual level and is excluded through aggregation. The performance was acceptable, with F1 measures at 60% (both for individual and aggregated results). High recall values (85%-87%) are important for EER verification to allow identifying as many defects as possible.

We also observed that open tasks lead to collecting IDRs not only at the focus me but also in its neighbourhood, tending to place defects at the level of entities and relations (rather than more fine-grained mes such as attributes and multiplicities) or even randomly. This finding could have an implication on the design of HC tasks to solve other similar model verification problems.

While they elicit broader insights from workers, the output of the open tasks is challenging to validate and aggregate automatically. In **RQ2**, we investigated to what extent a follow-up HC task could check IDR correctness and also assign these to predefined defect codes. Here we obtained good results for validation (F1 73%, 84%). As expected, the assignment of TD-ids is more difficult leading to lower performance results (F1 53%, 68%). In real-life settings, a TD Catalog might not be available and new approaches will be needed for aggregation (e.g., voting the best defect report).

Limitations and Threats to Validity. Internal validity focuses on descriptive statistics regarding the overall data sample. The authors introduced 35 defects in the *EER diagram* based on typical defects that occur in software development complemented with defects that have been identified during pilot runs. We applied a well-known application domain to avoid limitations caused by lack of domain knowledge. The experiment package was intensively reviewed by experts and tested in several pilot runs. *External validity* refers to the generalization of results. Participants were students from an academic course. Individual background experience has been collected in an experience questionnaire. As most participants work at least part-time in a professional software development environment, we consider them to be junior software developers. The main goal is to investigate *HC-based EER verification with open tasks* using a medium engineering model. However, the applied concepts are applicable for different domains and engineering models but might require considerable effort for setting up the defect verification and validation process with tool support. *Construct validity* focuses on the relationship between theory and observations (Wohlin et al. 2012). Applied measurements, such as defects reported, effort, P, R, and F1-score are well accepted in empirical studies. We applied a gold standard for seeded defects that has been extensively reviewed and used for evaluating reported defects. The study has been executed in class-room settings that hinders communication

and interaction between participants.

Future Work in the area of EER verification will focus on (i) in-depth analyses of the data sets with focus on a statistical analyses and hypotheses definition and testing, (ii) extending our analysis by taking into account worker skills from pre-experiment questionnaires; (iii) comparing results with those obtained with closed-task designs (Sabou et al. 2018b); (iv) investigating alternative task designs (e.g., granularity of focus entity, aggregation of IDRs); (v) working with larger models.On a longer term, research will focus on three areas. First, we see the need of further studies for other problem types and domains that would bring more insights in the benefits and draw-backs of open and closed HC tasks. Second, we are interested in extending our research to similar problem types, such as the verification of knowledge models (e.g., ontologies), to investigate how lessons learned can be generalized over problem types and domains. In particular, the verification of knowledge models such as ontologies is a promising application area where lessons learned in this work could be applied. Third, we aim at extending engineering tool chains with *Model Quality Assurance (MQA)* concepts for efficient and effective defect detection with HC. In a long term, we hope that our findings will contribute to making distributed, online work of information technology experts more efficient and effective.

8 Acknowledgements

We thank all students that took part in the experiments. M. Sabou was funded by the HOnEst Austrian Science Fund (FWF) project:V 745-N. Furthermore, we gratefully acknowledge the financial support of the Christian Doppler Research Association, the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development.

References

Acosta, M.; Zaveri, A.; Simperl, E.; Kontokostas, D.; Flöck, F.; and Lehmann, J. 2016. Detecting Linked Data Quality Issues via Crowdsourcing: A DBpedia Study. *Semantic Web Journal*.

Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; and Ives, Z. 2007. DBpedia: A Nucleus for a Web of Open Data. In *In 6th Int. Semantic Web Conference*, 11–15. Springer.

Aurum, A.; Petersson, H.; and Wohlin, C. 2002. State-of-the-art: software inspections after 25 years. *Software Testing, Verification and Reliability* 12(3):133–154.

Boehm, B., and Basili, V. R. 2005. Software defect reduction top 10 list. *Foundations of empirical software engineering: the legacy of Victor R. Basili* 426(37):426–431.

Eickhoff, C., and de Vries, A. P. 2013. Increasing cheat robustness of crowdsourcing tasks. *Information Retrieval* 16(2):121–137.

Hungerford, B. C.; Hevner, A. R.; and Collins, R. W. 2004. Reviewing software diagrams: A cognitive study. *IEEE Transactions on Software Engineering* 30(2):82–96.

Jain, A.; Das Sarma, A.; Parameswaran, A.; and Widom, J. 2017. Understanding workers, developing effective tasks, and enhancing marketplace dynamics: A study of a large crowdsourcing marketplace. In *Proceedings of the VLDB Endowment*, volume 10, 829–840.

Kollanus, S., and Koskinen, J. 2009. Survey of software inspection research. *The Open Software Engineering J.* 3(1).

Laitenberger, O., and DeBaud, J.-M. 2000. An encompassing life cycle centric survey of software inspection. *Journal of systems and software* 50(1):5–31.

Mao, K.; Capra, L.; Harman, M.; and Jia, Y. 2017. A survey of the use of crowdsourcing in software engineering. *Journal of Systems and Software* 126:57–84.

Mortensen, J. M.; Minty, E. P.; Januszyk, M.; Sweeney, T. E.; Rector, A. L.; Noy, N. F.; and Musen, M. A. 2015. Using the wisdom of the crowds to find critical errors in biomedical ontologies: a study of SNOMED CT. *Journal of American Medical Informatics (JAMIA)* 22(3):640–648.

Mortensen, J. M.; Telis, N.; Hughey, J. J.; Fan-Minogue, H.; Auken, K. V.; Dumontier, M.; and Musen, M. A. 2016. Is the crowd better as an assistant or a replacement in ontology engineering? an exploration through the lens of the gene ontology. *Journal of Biomedical Informatics* 60:199–209.

Moussawi, S., and Koufaris, M. 2013. The crowd on the assembly line: Designing tasks for a better crowdsourcing experience. In *International Conference on Information Systems (ICIS 2013)*, volume 4, 3745–3761.

Runeson, P. 2003. Using students as experiment subjects–an analysis on graduate and freshmen student data. In *7th Int. Conf. on Empirical Assessment in Softw. Eng.*, 95–102.

Sabou, M.; Aroyo, L.; Bozzon, A.; and Qarout, R. K. 2018a. Semantic Web and Human Computation: the Status of an Emerging Field. *Semantic Web* 9(3):1–12.

Sabou, M.; Winkler, D.; Penzerstadler, P.; and Biffl, S. 2018b. Verifying Conceptual Domain Models with Human Computation: A Case Study in Software Engineering. In *Sixth AAAI Conf. on Human Computation and Crowdsourcing*, 164–173.

Thalheim, B. 2009. Extended entity-relationship model. In *Encyclopedia of Database Systems*. Springer. 1083–1091.

Winkler, D.; Wimmer, M.; Berardinelli, L.; and Biffl, S. 2017. Towards model quality assurance for multi-disciplinary engineering. In Biffl, S.; Lüder, A.; and Gerhard, D., eds., *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, 433–457. Springer.

Wohlgenannt, G.; Sabou, M.; and Hanika, F. 2016. Crowd-based ontology engineering with the uComp Protégé plugin. *Semantic Web* 7(4):379–398.

Wohlin, C., and Aurum, A. 2004. An evaluation of checklist-based reading for entity-relationship diagrams. In *5th Int. Workshop on Enterprise Networking and Computing in Healthcare Industry*, 286–296. IEEE.

Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B.; and Wesslén, A. 2012. *Experimentation in software engineering*. Springer Science & Business Media.