FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Improving Crowdsourced Software Inspection: Development of an Experimental Process Support Platform

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Peter Penzenstadler, BSc.
Matrikelnummer 01127770

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Mag. Mag.rer.soc.oec. Dr.techn. Stefan Biffl
Mitwirkung: MSc., PhD Marta Reka Sabou

Wien, 8. Oktober 2018

Peter Penzenstadler	Stefan Biffl

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Improving Crowdsourced Software Inspection: Development of an Experimental Process Support Platform

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Peter Penzenstadler, BSc.
Registration Number 01127770

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Mag. Mag.rer.soc.oec. Dr.techn. Stefan Biffl
Assistance: MSc., PhD Marta Reka Sabou

Vienna, 8th October, 2018

Peter Penzenstadler | Stefan Biffl

Erklärung zur Verfassung der Arbeit

Peter Penzenstadler, BSc.
Herndlgasse 20/18, 1100 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 8. Oktober 2018

Peter Penzenstadler

Danksagung

An dieser Stelle möchte ich allen Menschen danken die mich während meines Studiums und dem Verfassen dieser Arbeit unterstützt haben.

Als erstes möchte ich meinem Betreuer Professor Stefan Biffl für die Möglichkeit danken diese Diplomarbeit zu schreiben. Ein großes Dankeschön ergeht an Marta Sabou, die mich während dieser Zeit, mit ihrer positive Einstellung und dem richtigen Feedback zur richtigen Zeit, begleitet hat.

Hier möchte ich mich auch Dankbar für meine Familie und alle meine geliebten Menschen zeigen, die mich auf diesem Weg begleitet haben. Ohne euch wäre das alles nicht möglich gewesen. Danke and meine Mutter, meinen Vater und meinem Bruder, dafür das ihr mir Stärke und Stabilität gegeben habt und mich bei jeder Entscheidung unterstützt habt. Danke an meine Freundin die mir in schwierigen Zeiten zur Seite gestanden ist und mir immer gesagt hat das ich das schaffe.

Danke an alle meine Freunde dafür das ihr mich immer daran erinnert habt das ich auf euch zählen kann wann immer ich euch brauche und das Leben mit Freunden mehr Spaß macht. Hier ist nicht genug Platz um meine Dankbarkeit für euch alle auszudrücken aber seid versichert, dass ich für jeden Menschen der mich auf dieser Reise begleitet hat Dankbar bin.

Acknowledgements

At this point I want to thank all the people who supported me during my studies and during the writing of this thesis.

First I want to thank my advisor Professor Stefan Bifflfor the opportunity to write this diploma thesis. A big thank you goes to Marta Sabou, for supporting me during this whole time with her positive attitude and by giving me the right feedback and advice when I needed it most.

This is also where I want to be grateful for my family and loved ones, who were with me all along this journey and without them this would not be possible. Thanks to my mother, father and brother, for always being there for me and giving me strength and stability during all this time and supporting me in every decision I make. Thanks to my girlfriend for standing by my side when times where tough and for ensuring me that I can do this.

Thanks to my friends for reminding me that I can always count on you in times when I need you and that life is also about enjoyment and company. There is not enough room to express my gratefulness to all of you, but be assured that I am thankful for everyone who walked with me on this journey.

Kurzfassung

Taxonomien und Wissensgraphen in Wissensmodellierung und Extended Entity Relationship (EER) Diagramme in der Software Entwicklung, sind Formen von Konzeptionellen Modellen und dienen als Basis zu Entwicklung von Informations-Systemen. Die Verifizierung dieser Konzeptionellen Modelle gegenüber ihrem Referenzsystem (Spezifikation) ist wesentlich um zu verhindern, dass defekte Model Elemente in den Software Entwicklungszyklus gelangen, da diese Modelle oft direkt zur Ableitung und Generierung von Software Artefakten, zum Beispiel in der Model-getriebenen Software Entwicklung, herangezogen werden.

Sabou, Winkler et. al, schlagen einen generischen Ansatz zur Verifizierung von Konzeptionellen Modellen (VeriCoM) vor, bei dem die traditionellen Software Inspizierungen mit Hilfe von Human Computation und Crowdsourcing erweitert werden. Die Performance von VeriCoM wurde mit Crowdsourced Software Inspection (CSI) Experimenten an einem Software Entwicklungs-Anwendungsfall getestet, bei dem die Korrektheit eines EER Diagramms in Bezug auf seine textuelle Spezifikation überprüft wurde. Diese Experimente folgten einem hauptsächlich manuell ausgeführten wissenschaftlichen Experiment Prozess, welcher sehr Zeit aufwändig, Fehler anfällig, nicht skalierbar und insgesamt nicht für größere Gruppen von Experiment-TeilnehmerInnen geeignet war. Daher entstand die Notwendigkeit für die Automation dieses Experiment Prozesses durch ein Software Programm.

Der Kern dieser Arbeit liegt in der Analyse des CSI-Experiment Prozesses und der detaillierten formalen Definition der benötigten Daten Modelle und Algorithmen zur Automatisierung von VeriCoM. Diese Daten Modelle und Algorithmen werden zur Entwicklung eines Prototypen einer CSI-Plattform herangezogen. Diese CSI-Plattform unterstützt die CSI-Experimente zur Validierung von VeriCoM. Während eines Tests des Prototypen unter Live-Bedingungen im Rahmen eines CSI-Experiments, wurde gezeigt, dass die entwickelte CSI-Plattform die Arbeitslast des Experiment Administrations-Teams stark reduzierte und die anschließende Umfrage zeigte zufriedene Experiment-beteiligte. Daher kann die Formale Definition von VeriCoM und der entwickelte CSI-Plattform Prototyp als Startpunkt für ein ausgereiftes Softwaresystem zur Automatisierung von VeriCoM und andere Experimenten innerhalb der CSI-Domäne herangezogen werden.

Schlüsselwörter

Crowdsourced Software Inspection, Human Computation & Crowdsourcing, Verifizierung von Konzeptionellen Modellen, Prozess Analyse, Formale Definition, Software Entwicklung

Abstract

Taxonomies and knowledge graphs in Knowledge Engineering and Extended Entity Relationship (EER) diagrams in Software Engineering, are forms of conceptual models and the basis for the development of information systems. Verifying these conceptual models against their frame of reference is crucial to prevent defective model elements from entering the Software Development Life Cycle, as these models can be directly used to create software artefacts and source code, by means of Model-Driven Software Engineering techniques. Enhancing traditional Software Inspection practices with Human Computation and Crowdsourcing, Sabou, Winkler et. al proposed a generic approach to Verify Conceptual Models (VeriCoM). The performance of VeriCoM was tested with Crowdsourced Software Inspection (CSI) experiments on a Software Engineering use case, verifying the correctness of an EER model with respect to a textual system specification. These experiments followed a mostly manually performed scientific experiment-process, which was very time-consuming, error prone, not scalable and overall not applicable for a larger crowd of participants. Thus, the need for automation of this experiment-process through a software tool arose.

The core work of this thesis consists of an analysis of the CSI-Experiment process and the detailed formal definition of the data model and algorithms needed to automate VeriCoM. These data models and algorithms are used to develop a CSI-Platform prototype which supports the experiments to validate VeriCoM. During a test under live conditions within a CSI-Experiment, the CSI-Platform greatly reduced the workload of the experiment administrations team and the subsequent evaluation questionnaire showed satisfied stakeholders. Therefore the formal definition of VeriCoM and the developed CSI-Platform can act as a starting point for the development of a full-fledged software system to automate VeriCoM and other experiments within CSI.

Keywords

Crowdsourced Software Inspection, Human Computation & Crowdsourcing, Verifying Conceptual Models, Process Analysis, Formal Definition, Software Engineering

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Problem description 1
1.2 Research Questions 5
1.3 Methodology 5
1.4 Contributions and Expected Results 8
1.5 Outline 9

2 Background and Related Work **11**
2.1 Software Inspection 11
2.2 Human Computation and Crowdsourcing (HC&C) 13
2.3 Crowdsourcing in Software Engineering (CSE) 13
2.4 Crowdsourced Software Inspection (CSI) 15
2.5 Crowdsourced Model Verification 16

3 CSI-Experiment Process Analysis & Requirements Elicitation **19**
3.1 CSI-Experiment Process Analysis 20
3.2 The VeriCoM Approach 22
3.3 CSI-Experiment Stakeholders 26
3.4 Requirement Analysis Method 27
3.5 Status Quo of the CSI-Experiment process 29
3.6 CSI-Platform Requirements 31

4 Data Model Design and Algorithm Development **35**
4.1 Data Model 35
4.2 Algorithms 45

5 Implementation **59**
5.1 Architecture and technology stack 59

5.2 VeriCoM - Algorithm implementation 62
5.3 Feedback Service - Module 68
5.4 User Interface 71

6 Evaluation of the CSI-Platform prototype **77**
6.1 Experiment Spring 2018 78
6.2 Comparison of unsupported experiment process vs. CSI-Platform supported process 79
6.3 CSI-Platform Evaluation - Interviews 82

7 Conclusion and Future Work **91**
7.1 Answers to Research Questions and Discussion 91
7.2 Limitations 93
7.3 Future work and Learning Analytics Platform (LEAP) 94

List of Figures **97**

List of Tables **99**

List of Algorithms **101**

Bibliography **103**

CHAPTER 1

Introduction

1.1 Problem description

Successful software development relies on well-elaborated system specifications and concisely formalised requirements as a foundation to craft conceptual models. These models can be directly used to create software artefacts and system parts, e.g. source code and test cases, by *Model-Driven Software Engineering* (*MDSE*) approaches [6]. Defects in these conceptual models have high impact on the derived code-base and on subsequent software development stages. To reduce the likelihood of defective model elements entering the *Software Development Life Cycle* (*SDLC*) and with that the source code, inspection methods need to be applied and the conceptual model needs to be verified. Software Inspection is a primarily human-based measure to check alignment of the designed conceptual software artefacts, e.g. Extended Entity Relationship (EER) models, with a corresponding frame of reference, e.g. specification documents such as requirement sheets and system specifications [3]. To inspect complex software models and overcome human limitations of cognitive focus, it is key to inspect small isolated components and provide them to a group of inspectors [17].

Based on traditional best-practice Software Inspection and enhanced by *Human Computation and Crowdsourcing* (*HC&C*) methods, Sabou, Winkler et al. presented the *Crowdsourced Software Inspection* (*CSI*) process [29]. CSI is an approach in the context of Software Engineering to inspect conceptual software models with the help of the expertise of an inspector crowd. During this inspection the inspectors check whether the conceptual model correctly and completely represents the description within a requirements document (i.e. frame of reference) [10]. Figure 1.1 illustrates the CSI-Process work-flow.

Starting the CSI-process is a preparation and planning phase (1), where reference documents and the corresponding software artefact (e.g. a conceptual model) are collected from the author and given to the CSI management team. In the following

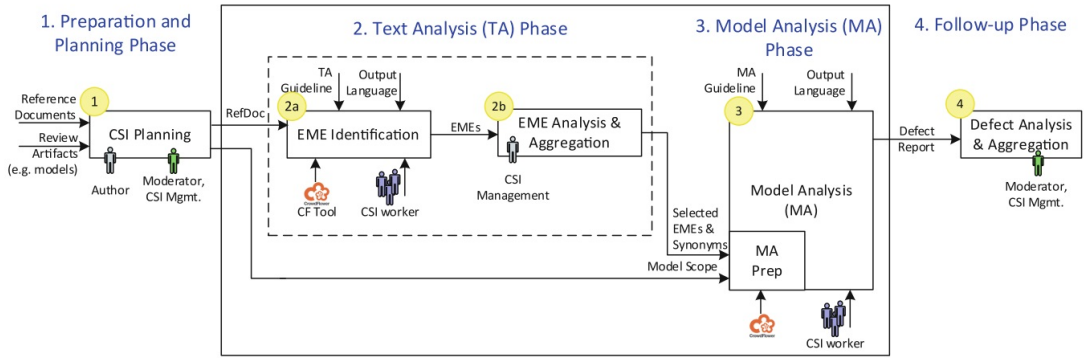


Figure 1.1: Crowdsourced Software Inspection (CSI) process [29].

inspection phase, the difference of CSI compared to the traditional inspection process, becomes clearer. There is a clear division between the *Text Analysis* phase (2) and the *Model Analysis* phase (3). During Text Analysis, the reference document is examined to identify *"Expected Model Elements" (EMEs)*, i.e. textually described concepts in the reference document, e.g. "A customer can order a menu item.", which should be correctly represented in the model. EMEs are then used as input for the Model Analysis phase and serve as reference to the specification and guidance for the workers performing the analysis. EMEs allow to split up the verification of a model into multiple small task that can be distributed to a crowd of workers which individually contribute to the verification of the whole model in a distributed and parallel fashion. For the Model Analysis (MA) phase, the workers receive a set of EMEs (one after the other) and check whether these are depicted correctly in the model. If they identify a erroneous representation they file a defect report describing the flaw. Output of the Model Analysis phase is a collection of defect reports which are aggregated and evaluated in the Follow-up phase(4). Finally this collection of defect reports should ideally identify all the defects contained in the model and can be used to guide the review and correction of the model.

Another form of conceptual models are ontologies, taxonomies and knowledge graphs, which represent the domain of the information systems in *Knowledge Engineering (KE)* [16]. Several knowledge management tasks of KE, and especially in Semantic Web, require human contribution [5]. To ensure correctness of ontologies, *ontology verification* is performed, which "compares the ontology against the ontology specification document (i.e. frame of reference)" [22].

The problem of verifying conceptual models shows high relevance in at least two research areas with different approaches to successfully tackle the problem in their specific field. Reaching into both Knowledge Engineering and Software Inspection to gather and analyse the differences and commonalities of these approaches, can lead to an exchange of experience between communities [23]. Thus, a generic approach to **Veri**fy **Co**nceptual **M**odels (**VeriCoM**) is introduced by Sabou, Winkler et al. [23]. VeriCoM offers a

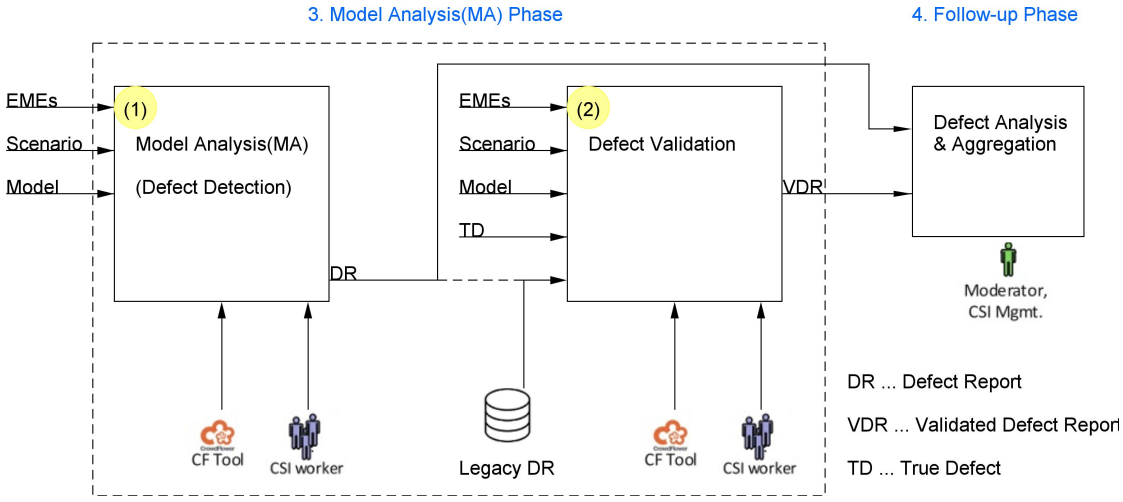
generic formalization of the problem, that can be applied across the fields of KE and SI.

Within the scope of a controlled experiment Sabou, Winkler et al. applied the generic VeriCoM-approach to a Software Engineering use case to test its applicability. CSI-Experiments were conducted under the VeriCoM-approach with the focus to evaluate the Model Analysis performance of an expert crowd, i.e., evaluation of defect detection effectiveness, efficiency, and false positives, compared to the traditional pen&paper Software Inspection approach [29]. To analyse VeriCoM, the Text Analysis phase is performed by the experiment administrators themselves to identify EMEs an to introduce gold standard defects (true defects - TD) into the model during the experiment setup. This experiment setup, illustrated in Figure 1.2a, also extends the original Model Analysis phase (see Figure 1.1 (3)) and introduces 2 subtasks, namely (1) Model Analysis (MA) and (2) Defect Validation (DV). In the Model Analysis task of the experiment, participants were presented a textual scenario description for context information, a corresponding EER model and a (expected) model element. Participants had to decide whether the model element was relevant and modelled correctly, or report a defect. The second subtask, Defect Validation (DV), was fed by legacy defect reports, scenarios and models from previous experiments together with true defect introduced by the experiment administration team. Goal was to match defect reports with true defects to validate their correctness. The resulting defect reports of VeriCoM have been compared to the output of the pen&paper approach. After aggregation and analysis of the defect reports by the CSI-Experiment team in the Follow-up phase, VeriCoM showed promising and comparable results.

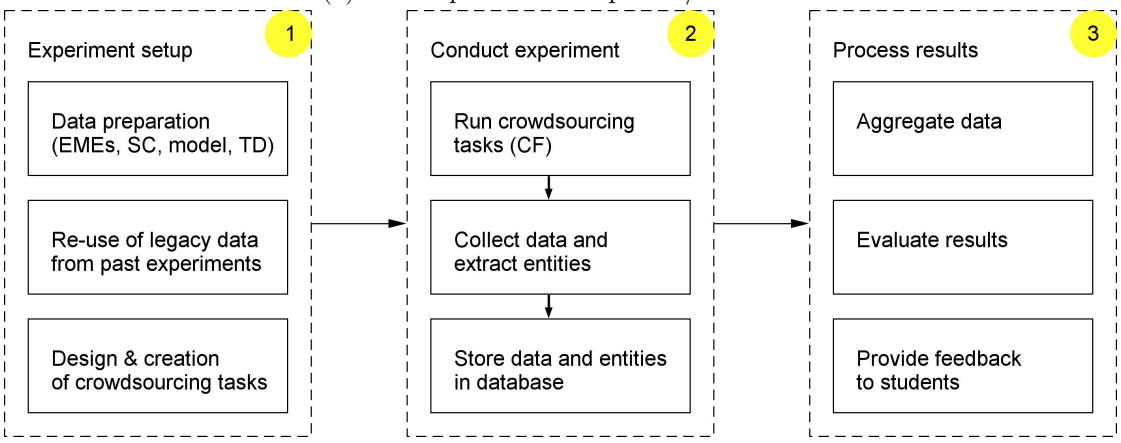
The experiments that validated VeriCoM, rely on an complex experimental process (see Figure 1.2b). This process consists of three consecutive steps. In the experiment setup step, the experimental data (EMEs, scenarios, conceptual models, etc.) together with the legacy data from previous experiments are selected and prepared. The experiments are then conducted with the help of the online crowdsourcing-tool FigureEight (FE) (f.k.a. CrowdFlower (CF)), which managed the (crowd-)workers, participating in the experiments. The selected experimental data is uploaded to FE and the crowdsourcing-jobs are created. FE schedules the crowdsourcing-jobs, distributes their tasks to the workers and collects their judgements together with corresponding meta-data (e.g. task-id, participant-id, trust-rating, etc.). After each workshop within an experiment round, the resulting data is collected and stored in a database. The experiment took place in a university setting, with students forming the expert-crowd and therefore also had an educational goal; to support students in developing model inspection skills. Therefore, in the last step, the resulting data is processed and evaluated, and students receive feedback on their model analysis performance through e-mail.

Major drawback of the described CSI-Experiment process is that it is mostly performed manually by the experiment administration team, with the exception of using FE for coordinating the expert crowd. The following shortcomings emerged:

- Handling of experiment data was cumbersome.



(a) CSI-Experiment setup 2017/autumn.



(b) Underlying experiment process.

Figure 1.2: CSI-Experiment setup and underlying process.

- Manual aggregation of results was inefficient.
- Manually conducted evaluation was time-consuming.
- Feedback for participants was moderate which restricted the benefit for the teaching purpose the experiments were intended to have.

Overall the CSI-Experiment process was slow, error-prone and not scalable. This hampers both further scientific investigations within this experiment family and potential industrial uptake. Another limiting factor is FE, which doesn't support the design and creation of task specifically tailored to the problem of verifying conceptual models.

Therefore, the core of this thesis was to thoroughly analyse the CSI-Experiments under VeriCoM, to elicit requirements for software development and to formally define the

data models and algorithms needed, to overcome the drawbacks of the current approach. Furthermore a prototype software tool, called the CSI-Platform, is designed and implemented, to support the CSI-Experiments and to test the developed data models and algorithms by automating cumbersome and time-consuming experiment tasks.

1.2 Research Questions

The research questions defined in this section, guide the analysis of the CSI-Experiment process and the development of the CSI-Platform.

- ***RQ1:*** *Which parts of the Crowdsourced Software Inspection experiment process benefit the most from software tool support?* The CSI-Experiment process will be thoroughly analysed and parts which are in need of software tool support will be identified.
- ***RQ2:*** *What are the requirements for the software platform?* To meet the key user (i.e. experiment administrators) expectations for the platform, requirements need to be defined that represent the key users needs.
- ***RQ3:*** *How can the CSI domain model be defined in a generic way?* The domain model of the CSI-Experiment process and the supporting platform need to be defined in a generic way in order to allow for the extension of the platform, the connection to different crowdsourcing-engines (other than FE) and the creation of a Learning Analytics Platform (LEAP, see Section 7.3) planned in the future.
- ***RQ4:*** *Did the developed platform meet the expectations of the experiment administration team?* Validation and verification of the developed platform is key to learn whether or not the intended users are accepting the presented solution.

1.3 Methodology

Apart from literature research the methodological approach follows a Software Development Life Cycle (SDLC) [20].

- **M1: Literature research**: To get a clear overview of the current state of *Software Quality Assurance and Inspection* methods, a literature research was conducted. Starting with the basics of Software Inspection (SI), on to Human Computation and Crowdsourcing (HC&C) and its current applications in software engineering (Crowdsourcing in Software Engineering - CSE), to finally investigating current appliances of HC&C methods to SI in the Crowdsourced Software Inspections (CSI). As the field of CSI is considered a novel approach in software model inspection, the experiments of Winkler, Sabou et al. [24] will act as a foundation for this work.

- **M2: Process Analysis & Requirements Engineering**: As a groundwork for improving the CSI experiment process, a thorough understanding of the study procedure and its individual steps is required to identify candidates which would benefit the most from software tool support. Therefore a comprehensive analysis of the currently applied CSI-Experiment process will be conducted. For each of the candidate process steps a requirements elicitation will be performed to derive functional and non-functional requirements for the software prototype. These requirements will then be translated into design artefacts and features of the developed software platform.

- **M3: Model design**: In the design phase of this work, the domain model of the CSI-Experiment process is formalized and the key concepts are defined and described in detail. The existing data models, artefacts and concepts designed for the expert sourcing experiment, described in [24], are well developed and sufficient for a scientific evaluation of CSI performance. However, some adaptations, extensions and improvements need to be made to allow for a well-designed software architecture of the supporting platform and the automation of certain process stages. This redesign will include the introduction of typed objects, adaptable interfaces for existing and newly created domain entities and services, a generic interface to connect to crowdsourcing-engines and enable automated evaluation and feedback for experiment participants. The design phase is directly linked to the implementation phase of the thesis.

- **M4: Implementation**: Taking the resulting new design artefacts and data models of the previous step, a prototypical implementation of a CSI-Platform will be presented. The focus of this prototype will lie on supporting the moderator before, during and after conduction of experiments, through improved management of experiment data and automated result evaluation. It should also be capable of connecting to a variety of crowd sourcing engines and therefore be independent of FE. To speed up the development process a Spring Boot based web application generator (JHipster[1] [19]) will be used, which derives a software scaffold from the introduced data models and allows to focus on crowdsourcing service API integration, outcome evaluation and feedback provision, by reducing tool configuration effort to a minimum. The utilization of this tool is also a showcase for the usage of model-driven engineering techniques in practice, as it derives a completely usable software prototype from a designed EER-diagram.

- **M5: Testing**: To test the implemented prototype, it will be deployed during a CSI-Experiment under live conditions. The experiment will be held as part of the university course of "Software Quality Assurance" and will be conducted by Sabou & Winkler in a similar setup as described in [24] with student experts. This test run should prove the prototype's applicability in a real test environment and will act as a foundation for the evaluation phase of this thesis.

[1] JHipster: http://www.jhipster.tech/

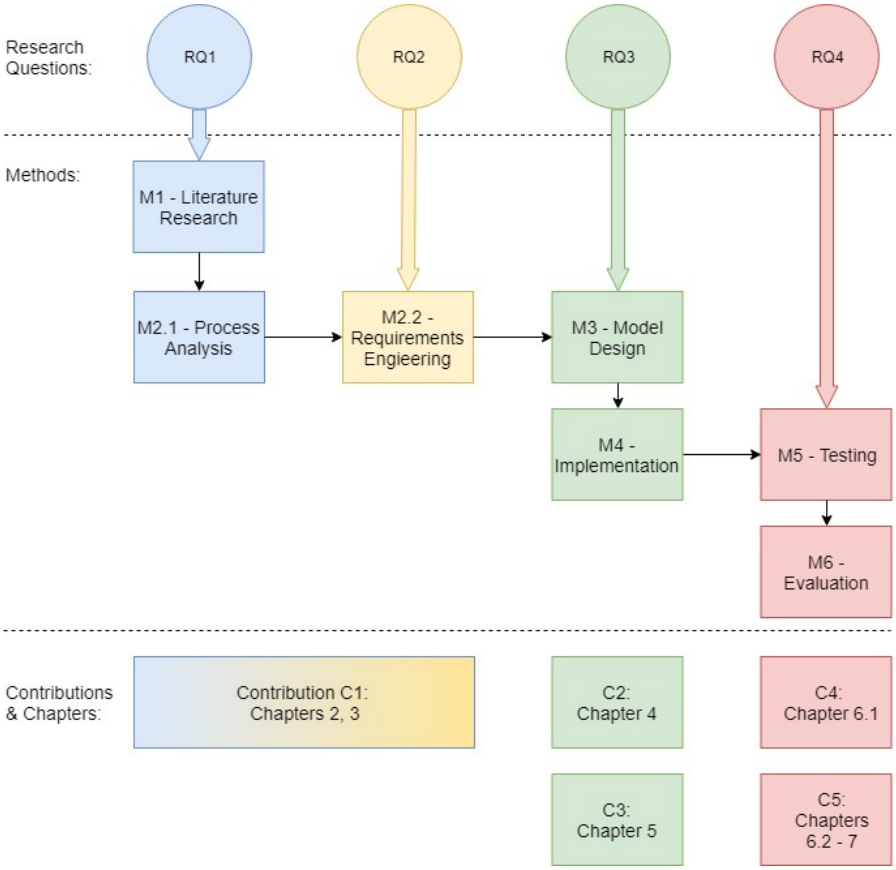


Figure 1.3: Research questions aligned with methods, contributions and chapters.

- **M6: Evaluation**: Based on the outcome of the planned experimental application of the new prototype, a detailed comparison between experiment execution with and without the supporting platform will be drawn. The evaluation should clarify which parts of the process benefited from tool support in regards to improvements in speed of evaluation, efficiency, scalability and error reduction. Also novel features, enabled by the prototype, e.g. immediate student feedback, will be evaluated. To gather feedback from the experiment administration team, a workshop, including a questionnaire to gather improvement suggestions, will be held after the experiment to present the experiment-support offered by the software tool to all the people involved during the conducted experiment.

By following this methodology the impacts of the CSI-Platform on the CSI-Experiment process can be observed.

Figure 1.3 aligns the research questions (RQx) with the performed methodology to address the specific question and shows the contributions made, as well as the chapters of the thesis where they are described.

1.4 Contributions and Expected Results

Overall goal of the thesis is to support and improve the CSI experimental process. The following results are expected (Cx - Contribution):

- **C1: Analysis of the CSI-Experiment process and requirements elicitation**: To understand how the CSI experiment process can be improved, a thorough analysis of each process step has to be performed, evaluating their potential for improvement and automation. Based on the outcomes of the analysis, a set of functional and non-functional requirements for the software prototype should be derived. These requirements should cover the needs of the main stakeholders of the experiment process (i.e. the experiment management team) and act as a starting point for the software design and implementation phase. Another result of this analysis phase should be a clear definition and vision for the software platform that is to be developed and also highlight potential future extensions of the platform.

- **C2: Design of models and artefacts for the prototype solution**: A complete formal definition of VeriCoM and its application to the software engineering use case is desired, together with the creation of all need objects and types from the CSI domain. Based on the findings during process analysis and requirements definition, artefacts, which represent the desired software platform in an abstract way, have to be designed. The existing experiment context and data model definitions this thesis builds upon, have been designed specifically for the FigureEight crowdsourcing service. Therefore, to become applicable in industry and to allow for connection to different crowdsourcing platforms in future experiments, CSI-process data structures for model analysis and defect validation have to be generalized to generic model structures. These should be detached from the FE environment and adaptable to other crowdsourcing engines. Additionally, new elements and services for an automated evaluation of the experiment results are part of this design.

- **C3: Prototypical implementation of the CSI-Platform for supporting CSI-Experiments**: Practical outcome of this thesis is the implementation of a CSI-Platform, which enhances various steps of the CSI-Experiment process through automation. The web application should aid with conducting CSI-Experiments, like the ones performed in [29, 24]. The platform should support the experiment process shown in Figure 1.1 by providing services to manage input data for the crowdsourcing jobs (1), triggering crowdsourcing experiments and automatic collection of output data (2), statistical evaluation methods and automated result analysis for experiment administrators, as well as providing a personalized e-mail report to the student-crowd, which is solving the crowdsourcing tasks (3). Furthermore, comprehensive performance information on overall group output, efficiency, accuracy and agreement on defect verification between the workers is measured and presented by the platform.
  Generation of experiment input data will not be in the scope of this thesis.

- **C4: Support CSI-Experiments with automated evaluation**: During the preparation for this thesis, the second round of CSI-Experiments has already been conducted by Winkler, Sabou et al.[24]. While the evaluation of the first experiment round was done manually, which took a lot of effort and time, the second round already benefited from a proof of concept (PoC) implementation of a supporting evaluation platform. This PoC has already been developed by the author specifically for the CSI-Experiment in fall of 2017 and showed promising results. Goal of this work is to further expand the capabilities of this PoC and provide a full-fledged experiment support tool for future experiments, which are already planned within this experiment-family.
- **C5: Evaluation of improvements to the CSI-Experiment process enabled by the CSI-Platform**: Experiment process parts, which are candidates for improvement through automation (identified during the analysis), will be evaluated in terms of efficiency and effectiveness and compared to how work on the respective part was conducted prior to the development of the supporting software platform. Being part of this evaluation is the capturing of the satisfaction level of the experiment administrators with the developed platform. Also a list of suggestions for improvements and possible requirements for planned future work should be outcome of this evaluation.

1.5 Outline

The rest of the thesis is structured in the following way. Chapter 2 describes the current state of the art of Verifying Conceptual Models (VeriCoM) and its background rooted in Software Inspection as well as Human Computation and Crowdsourcing. A detailed analysis of the CSI-Experiment process and the VeriCoM-approach, followed by a subsequent requirements elicitation for the CSI-Platform, is performed in Chapter 3. The identified requirements for the CSI-Platform are then used to design the data models and algorithms, which are described in Chapter 4. These data models are then used in Chapter 5 for the concrete implementation of the CSI-Platform prototype. The developed prototype was then tested during the CSI-Experiment in spring 2018 and a comparison to previous experiment runs is drawn in Chapter 6, together with an evaluation of the satisfaction of the experiment management with the developed CSI-Platform. Chapter 7 draws a conclusion to the thesis and gives suggestions for future work.

CHAPTER 2

Background and Related Work

This chapter covers the background of Crowd-sourced Software Inspection (CSI) and places it within the field of Software Inspection. Section 2.1 describes the origin of Software Inspection and the original approach proposed by Michael Fagan. Section 2.2 defines the terms Human Computation and Crowdsourcing and addresses their relationship. Applications of Crowdsourcing within the Software Engineering context is described in 2.3. The application of Crowdsourcing techniques to perform Software Inspection is detailed in Section 2.4, while Section 2.5 addresses the field Crowdsourced Model Verification.

2.1 Software Inspection

Software Inspections (*SI*), originally developed by Michael Fagan in 1976, are a systematic approach to examine a program or software design artefact in detail [17]. They are tools to verify conceptual design models and to obtain and improve code quality through a guided verification process with well-defined roles for inspection practitioners [7]. This structured approach to software inspection has been well accepted by the software development industry as the key methods to assure the creation of high quality software artefacts [3]. The minimal group of key roles in this approach are moderators, which lead and manage the inspection and a group of inspectors, which perform the inspection tasks [3, 7]. Goal of software inspection is to verify correctness, completeness and conformance of the developed product to its corresponding requirements and specification documents. Inspection methods should already be applied in the design phase of the program on the conceptual models, to assure a concise foundation for writing the program code.

The traditional software inspection process proposed by Fagan [7], consists of five steps:

1. Inspection Planning: a structured and well planned approach to inspect a piece of software, leads to a repeatable process that can be incorporated into routinely

applied quality assurance measures by the development team. Therefore the inspection team needs to be composed, the software part to inspect needs to be prepared and the inspection needs to be scheduled.

2. Individual Defect Detection: Each team member inspects the prepared software piece and its related conceptual model artefacts individually and tries to identify defects.

3. Team meetings: In the planning phase, team meetings need to be scheduled to collect, discuss and aggregate individual defect reports.

4. Rework by the authors: With all the individual defect reports aggregated, the respective authors of the piece of software get presented the collection of the identified defects, to rework and re-factor their code to mitigate the defects.

5. Inspection Closure: After the defects have been fixed, the inspection has to be reviewed, the improvements are archived and the inspection is closed by the team.

Software Inspection faces several challenges that need to be overcome to ensure that the software models meet the desired level of quality [27].
As software projects grow larger and conceptual domain models become more complex, the pressure on reviewers increases as smaller details become harder to inspect while inspection time increases. This becomes more immanent, as researchers agree that a slower review process yields better results, with a suggested optimum reviewing rate of one page per hour per participant [21, 8]. Therefore a policy of *divide and conquer* compensates for limitations on review time frame and handling details of large conceptual models [17]. By having each inspector examine small isolated components, chances are increased, that a) nothing is overlooked and b) that the correctness of each component implies the correctness of the whole product [3].
The second challenge for the quality assurance (software inspection) team of a software project is, that there are limited ways of cost-effective coordination of the inspection teams [27].
Third, while best-practice reading techniques systematically guide the inspection team through textual specification and code, there is only limited available support for inspecting design models.
Those challenges underlines the need for software tool support to manage the inspection teams and to guide them through the inspection of models, in order to achieve systematic coverage of large specification documents and conceptual models [30].

Fagan states that the manner in which inspection data is categorized and presented in the inspection process is an important factor to attain improvements and can lead to initial and ever-improving error reduction [7]. Parnas et al. also state that customizing the inspection process can provide benefits, therefore enhancing SI with Human Computation and Crowdsourcing methods promises to be a reasonable approach [17].

2.2 Human Computation and Crowdsourcing (HC&C)

The terms *Human Computation and Crowdsourcing (HC&C)* are often used interchangeably, despite the fact that they describe two different paradigms with a shared overlap in their fields of use [18]. *Human Computation* can be defined as: *"... a paradigm for utilizing human processing power to solve problem that computer cannot yet solve."* [28]. Whereas crowdsourcing redirects a job traditionally performed by a designated worker (e.g. employee of a company, researcher etc.) to an outsourced, undefined and generally large group of people in an open call format [9]. In other words, *Human Computation* replaces computers with humans, while crowdsourcing replaces traditional human workers with members of the public [18].

Application of HC&C techniques, allow for large complex problems to be split up into smaller, more easily solvable tasks, which can be distributed to a suitable population of contributors and enable coordination and aggregation of the individually resolved tasks [30]. Therefore HC&C techniques are applied in a wide variety of use-cases such as object recognition, object classification and image interpretation. These are examples of tasks where humans still out-perform computers.

As software specifications are often written by people without expertise in software development, the specification text is most of the time written in natural language that vaguely describes the intended software use case, it is hard to be interpreted automatically by algorithms. Also, knowledge about the field of use is required to correctly inspect the textual specification with respect to the modelled software artefact. Adapting an algorithm to fit one use case (e.g. a booking service for a restaurant) is very time consuming and of limited use if the domain changes. Human workers (e.g. a group of experts) on the other hand, excel in being able to adapt to changing domains and understanding the desired functionality that lies behind the specification and the respective domain model. Therefore they out-perform algorithms by consistently finding defects in abstract descriptions and models.
Inspection of larger specification documents and software artefact by single inspectors however, can lead to fatigue and a reduction of effective defect detection. To compensate for this limitations of human inspectors, HC&C techniques can be applied to distribute the workload onto multiple inspectors.

Enriching Software Inspection with HC&C mechanisms therefore seems to be a promising approach, to reduce cognitive fatigue of single inspectors, improve coverage of large models and speed up the inspection process by parallelization of tasks [30].

2.3 Crowdsourcing in Software Engineering (CSE)

The advent of micro tasking platforms such as *Amazon Mechanical Turk* [1] (AMT) or *FigureEight*[2] (FE) (f.k.a. *CrowdFlower*(CF)) and their usage within the Software

[1] Amazon Mechanical Turk: https://www.mturk.com/
[2] FigureEight: https://www.figure-eight.com/

Engineering (SE) context have helped the novel research area of Crowdsourced Software Engineering (CSE) to surface [24]. CSE is 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" [11, 12]. Three different models of CSE can be distinct [11]:

- Peer production: Open source development is the most prominent example of peer production. Tens of thousands of people contribute to the development of a collectively scoped software projects under common goals, in order to gain experience and reputation within the SE community. Linux [3], Apache, Rails and Firefox [4] are examples for successful crowd developed software projects. Also StackOverflow [5], the most prominent forum for developers to share their expertise, can be counted as a peer produced knowledge base for software development.
- Competitions: Competitions recently became very popular in software development and are similar to traditional outsourcing. A client defines and requests work from platforms like TopCoder [6] (i.e. platform for rapid software prototypes) or 99designs [7] (i.e. crowd-sourced visual designs), an receives results from the winner(s) of a competing crowd. Competitions are managed by a co-pilot which decomposes the request into multiple competitive task and selects the winning entry.
- Microtasking: This model distinguishes itself by its high scalability. Large complex tasks are split into sets of self-contained microtasks, which can be quickly solved by a large group of distributed workers and then composed into a solution for the original task. Quality is insured by asking multiple workers to complete the same task and selecting the solution voted the highest.

Choosing the right model for a given project depends in different properties of the project and the desired results, e.g. needed crowd size (e.g. small, large), time to solve each (micro)task (e.g. minutes, days), required contributor expertise, incentive mechanism used motivate contribution (intrinsic, extrinsic) or the needed context to solve a task(none, extensive) [30].

The CSE community shows increasing interest in applying crowdsourcing and micro tasking techniques in all phases of the software development life cycle (*SDLC*) and contributions to the phases planning and analysis (17 papers), design (4 papers), implementation (26 papers), testing (22 papers), and maintenance (26 papers) could be observed by a recent survey from Mao et al. [12]. Software inspection, however, is weakly addressed, which is why this work focuses on this aspect of software development. Benefits of CSE and its application in Software Inspection in comparison to traditional inspection methods are, e.g. increased scalability, improved coordination and control, increased

[3] Linux: https://www.linuxfoundation.org/projects/
[4] Firefox: https://developer.mozilla.org/en-US/
[5] StackOverflow: https://stackoverflow.com/
[6] TopCoder: https://www.topcoder.com/
[7] 99designs: https://www.99designs.com/

coverage and the inclusion of non-necessarily expert contributors, which perform well defined small task units in parallel to accelerate the inspection process [29].

2.4 Crowdsourced Software Inspection (CSI)

Crowdsourced Software Inspection adapts the traditional Software Inspection process and extends it with the capabilities which are offered by crowdsourcing platforms, e.g. FigureEight, to introduce human computation mechanisms to the traditional inspection process. Special focus was put on the *Preparation* and *Software Inspection* phases (i.e., inspection planning, individual defect detection and team meeting) of the original SI approach introduced by Fagan et al. [29]. These phases were customized in order to allow for the application of crowdsourcing techniques. Goal of CSI is, to be able to split up complex software models, which are hard to analyse and demand a high amount of cognitive focus, into small inspection tasks of isolated components, that can be distribute and solved by a possibly large group of inspectors.

To be able to split up the task of analysing a conceptual model into isolated components, the model needs to be dissected into the elements which make up the model, called model elements (ME). These MEs represent entities, entity attributes, relationships, and relationship attributes from the model (e.g. an EER model [26]). Hereby *Expected Model Elements (EMEs)* are a concept introduced by Winkler et al. as: "key concepts that can be expected in the model under inspection based on inputs from reference documents, i.e., software requirements and specifications" [30]. EMEs are therefore derived from the frame of reference, i.e. the specification document.

The CSI-Process consists of four phases (see Figure 1.1):

1. Preparation and Planning phase: Reference documents and review artefacts, e.g. the textual specification and EER model, are prepared, the scope of the inspection is defined and the crowdsourcing environment is set up by the CSI management.

2. Text Analysis phase: The reference document is analysed and EMEs are derived.

3. Model Analysis phase: The identified EMEs are used for inspection of the model and found flaws are filed as a defect reported.

4. Follow-up phase: Defect reports are aggregated and evaluated and are used to later review and fix the identified defects.

In the Model Analysis (3) phase, EMEs are presented to the CSI crowd-workers which individually contribute to the verification of the whole model in a distributed and parallel fashion. They are tasked to judge, whether the representation of the EME in the designed model is correct. After the CSI workers completed their tasks, the judgements are aggregated and analysed by the CSI management team in the Follow-up phase (4).

The CSI management benefits from improved coordination and control over the inspection team members, tasks and results, an increased coverage of potentially large and diverse artefacts, as well as an accelerated inspection process due to the parallelization of small tasks [29]. The main problem however, is the manual work performed when aggregating and evaluating the defect reports in the Follow-up phase of the process. Infrastructure and automation through software to support the CSI process are lacking, which, if developed, would make experiments within this experiment family faster, less cumbersome and easier to run and would allow for industry adoption of the CSI-process.

2.4.1 FigureEight (f.k.a CrowdFlower)

The crowdsourcing engine FigureEight is used to manage the crowd of workers participating in the experiments to verify VeriCoM. The platform allows the creating of crowdsourcing jobs of different predefined types, e.g. Sentiment Analysis, Search Relevance, Data Categorization, Data Collection & Enrichment, Data Validation, Image Annotation, Transcription, Content Moderation and also offers tools to create custom jobs. This custom job creation tool was used to create the Model Analysis and Defect Validation jobs for the CSI-Experiments.
Besides job creation, FigureEight automatically manages the distribution of tasks to workers, the scheduling of jobs, the collection of judgements of workers, offers job monitoring and different statistical evaluation methods regarding the analysis of contributions. It also adds a "trust"-rating to workers, indicating how trustworthy a workers judgements on his tasks are.
FigureEight is not a free service and has increasing costs based on the supported worker crowd and the amount of judgements that can be gathered. This is a major drawback when using the service in a CSI-Experiment. It also limits the applicability within a university context and therefore a replacement for FigureEight has to be found or developed.

2.5 Crowdsourced Model Verification

Verification of conceptual models by utilizing human computation capabilities of both, field experts and layman crowds, has been part of research efforts across the fields of Knowledge Engineering and Software Engineering. Related work exploring and successfully applying Crowdsourced Model Verification is shown in Table 2.1 and discussed in this section [23].

In [1], Acosta et al. compare the contributions of experts and a layman crowd on evaluating the quality of triples from *Linked Data Knowledge Graphs*. The goal is to identify quality issues appearing frequently in *DBpedia* ([2]) triples. Experts were allowed to choose between three strategies of data selection when starting to work on their expert-sourcing task: (a) random suggestion of data, (b) data from a selected class, or (c) manually selected data. One predefined quality issue, picked from a taxonomy of issues, was assigned to the evaluated triple. Crowdsourcing was performed in a two staged experiment with randomly selected data. First one of three possible issues (i.e.,

Table 2.1: Overview of related work.

Paper	**Evaluated Elements**	**Frame of Reference**	**Defect Types**
Acosta et al. 2016 - Expert [1]	Data Triples	Human Knowledge	Defect Taxonomy
Acosta et al. 2016 - Crowd [1]	Data Triples	Human Knowledge	3 or 2 defects
Mortensen et al. 2015; 2016 [13, 14]	Subsumption Relations	Human Knowledge	Binary
Wohlgenannt et al. 2016 [32]	Terms, Relations	Human Knowledge	Binary
Sun et al. 2016 [25]	Taxonomy	N/A	N/A
Winkler et al. 2017 [30]	EER Model	System Specification	Open ended
Sabou et al. 2018 [23]	EER Model	System Specification	Multiple Defect Types

value, link, datatype) is selected by the workers, then in the second stage, workers solve dedicated jobs for individual quality issues and specify whether a corresponding triple is correct/incorrect. Verifying the correctness of subsumption relations in large, domain specific ontologies such as *SNOMED* and the *Gene Ontology*, Mortensen et al. report on their findings on crowd-sourced ontology verification with the help of CrowdFlower. Tasks for the workers contain the relation that has to be evaluated for correctness and a field to provide an explanation of their choice. The authors experiment showed that crowds have a comparable agreement rate with a baseline expert evaluation for SNOMED and can act as a scalable assistance in ontology engineering. Wohlgenannt et al. aimed to bring crowdsourcing closer to ontology engineers by developing an extending plugin for the Protégé ontology editor ([15]) [32]. Sun et al. [25] aimed to evaluate taxonomies in terms of how well they support user navigation [23]. In the field of Software Engineering, the works on the verification of conceptual models is limited to the works of Sabou, Winkler et al. [23, 30], in which they evaluate EER diagrams with respect to a textual specification document. During the first experiments, workers provided free-text descriptions of defect, which led to overly complex aggregation and the later introduction of defect types.

2.5.1 VeriCoM approach

An application of Crowdsourced Model Verification is the verification of conceptual domain models with respect to a textual specification. Therefore Sabou et al. introduced the generic *Verifying Conceptual Models (VeriCoM)* approach, described in detail in Section 3.2. VeriCoM is a generic approach that can be applied to a variety of Model Verification tasks. In [23] VeriCoM is applied to a Software Engineering (SE) use case

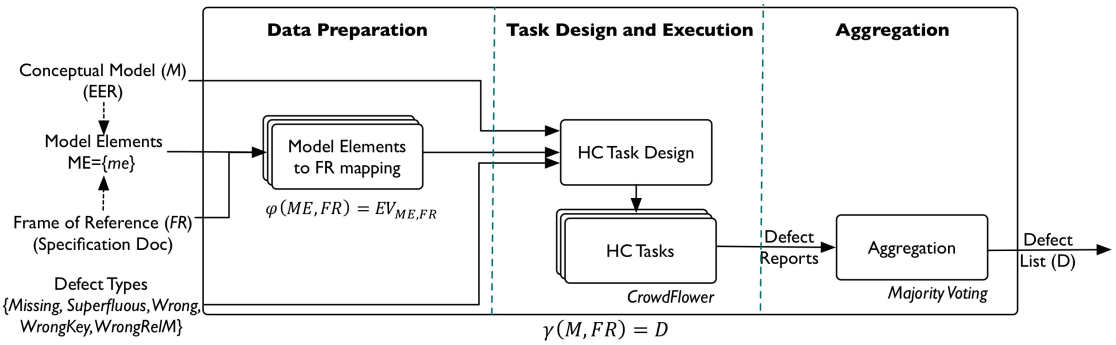


Figure 2.1: VeriCoM applied to a SE use case [23].

with an EER diagram as the conceptual model and a specification document as the frame of reference. Figure 2.1 described the process of the conducted experiments in paper [23] and its phases. Text Analysis was performed by the CSI-Experiment administration team, while the focus of the experiment was to analyse the performance of the Model Analysis phase of VeriCoM, compared with the traditional pen & paper approach. After Text Analysis, i.e. identification of Model Elements and definition of Defect Types, the input data was prepared and mapped to their corresponding part of the specification document, called Scenario. These Scenarios together with their ME and the EER model were used to create and execute the Human Computation (HC) tasks. These HC tasks were performed by the crowd workers, which filed a set of Defect Reports for each defect identified during Model Analysis. In the next phase these Defect Reports were aggregated into a list of defects and evaluated by the CSI-Experiment administration team. The steps of VeriCoM and a formal definition of all the variables and functions of Figure 2.1 can be found in Section 3.2.

CHAPTER 3

CSI-Experiment Process Analysis & Requirements Elicitation

Addressing Research Questions RQ1 and RQ2 lays the foundation for improving the CSI-Experiment process (see Figure 1.1). To identify requirements which improve the process, a thorough understanding of the study's procedure and its individual steps is needed. Therefore an analysis of the CSI-Experiment process is performed (see Section 3.1). The CSI-Experiment process is applied to test an approach to **Veri**fy **Co**nceptual **M**odels (**VeriCoM**) proposed in [23] by Sabou et. al. and described further in Section 3.2. VeriCoM is conceptualized in a generic way and is therefore applicable in a variety of model inspection use cases. In [23] the VeriCoM approach is applied to a Software Engineering (SE) use case, where, given a textual specification document, an EER model is checked for correctness and completeness. This SE use case forms the basis of the CSI-Experiment and the developed CSI-Platform described in this paper.

The stakeholders of the envisioned platform and their different interests and desired functionalities for such a platform are defined in Section 3.3. Section 3.4 describes how process candidates in need of tool support have been detected through workshops with experiment administrators and involvement in the running of the experiments in autumn 2017. The status quo of the CSI-Experiment process was examined and major issues and drawbacks of the unsupported process steps are formalized in section 3.5. Based upon these drawback, parts in need of automation and software tool support have been identified and requirements for the CIS-Platform are derived in Section 3.6.

Outcome of this process analysis and requirements elicitation is a specification for the CSI-Platform, which is used to support experiments performed with the VeriCoM approach. This specification of functional and non-functional requirements is later used for the design 4 and implementation 5 of the CIS-Platform.

Table 3.1: Conducted experiments to verify VeriCoM.

Experiment	Tasks	Description
Exp. 2016 Autumn	Text Analysis, Model Analysis (free text)	Identification of model elements and Model Analysis with free text description of defects.
Exp. 2017 Spring	Text Analysis, Model Analysis (free text)	Identification of model elements and Model Analysis with free text description of defects.
Exp. 2017 Autumn, Exp. 2018 Spring	Model Analysis (defect types), Defect Validation	Model Analysis with guiding questions and defect types. Validation of defects from previous experiments.

3.1 CSI-Experiment Process Analysis

During the course of the development of VeriCoM, several rounds of experiments have been performed to verify the crowd-sourced inspection approach (see Table 3.1). The experiments have been gradually improved with each experiment round. Up until spring 2017 the tasks performed in the experiments included Text Analysis (identification of model elements in the specification text) and Model Analysis (verification of a model diagram) with a free text description of defect reports. Because these free text descriptions were hard to analyse and required manual work, defect types were introduced and the Defect Validation task replaced Text Analysis to be able to align legacy defect reports to the new defect types. Section 3.1.1 described the experiment performed in Autumn 2017, which marked the entry point for the development of the CSI-Platform.

3.1.1 CSI-Experiment: Experiment 2017 Autumn

In autumn of 2017 an experiment was conducted to evaluate the VeriCoM approach. The experiments should (a) investigate the feasibility of VeriCoM and (b) compare the defect detection performance of this new process to the traditional best-practice pen-and-paper (P&P) software inspection process [29].
During the study preparation the study material (i.e., textual specification, FE configuration, task guidance tutorials and questionnaires) has been organized and the crowdsourcing engine FE has been set up. The overall experiment was set in a university context and was split up into 4 workshops to allow for a better coordination of the participating students and to mitigate the spacial constraints of the university infrastructure.
For each workshop, the group of participants was split in to three groups i.e., A, B and C. Groups A and B performed Model Analysis applying VeriCoM, while C applied the traditional P&P process and functioned as a control group. Before the Model Analysis phase, each participant had to fill out a online questionnaire, capturing the previous experience of the participants and other relevant data e.g., e-mail addresses to send the feedback mails to. Also a 30 min. tutorial explained the crowdsourcing tool and the

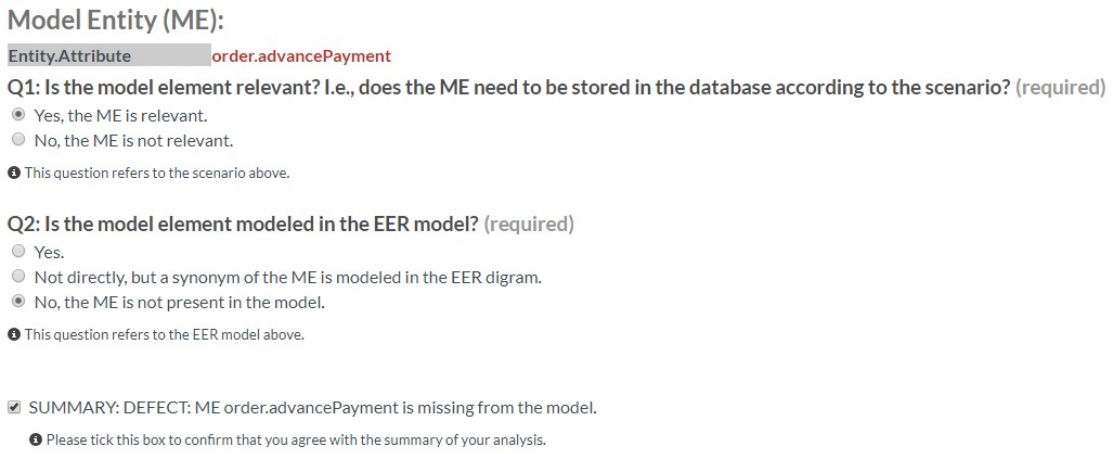


Figure 3.1: Guiding questions for the Model Analysis task.

tasks ahead.

Model Analysis

During the Model Analysis (a.k.a. Model Verification, Defect Detection) task, the participants had to analyse the conceptual model with respect to the textual specification and identify defects.
The input data for this task was split into 12 batches with 10 emes per batch and for each batch a FigureEight job was created. The created jobs contained the same context information (i.e., a restaurant specification), but focused on different parts of the specification called scenarios (e.g., accepting of orders, buying of ingredients, etc.).
Each student of group A and B was assigned to three jobs at random. During each job, the students had to answer the guiding questions to the presented model element and file defect reports (see Figure 3.1). Overall the students had a maximum of 60 minutes to finish the three jobs.

Defect Validation

The Defect Validation task was feed by legacy defect reports from precious experiments. This task was created to incorporate the legacy defect reports, i.e. free-text defect reports, into the new definition of defect reports (see Section 4.1.5), by matching them with predefined TrueDefects (i.e., validating them with respect to the gold standard defects) and assigning them the same defect type as the TrueDefect they matched with. This allows for them to be treated in the same way as the defect reports filed in the new experiment rounds.
Therefore, the students got presented, the model element (ME) for which the defect report was filed and the textual description of the defect report. By answering the guiding questions (see Figure 3.2), the students had to first validate the defect report,

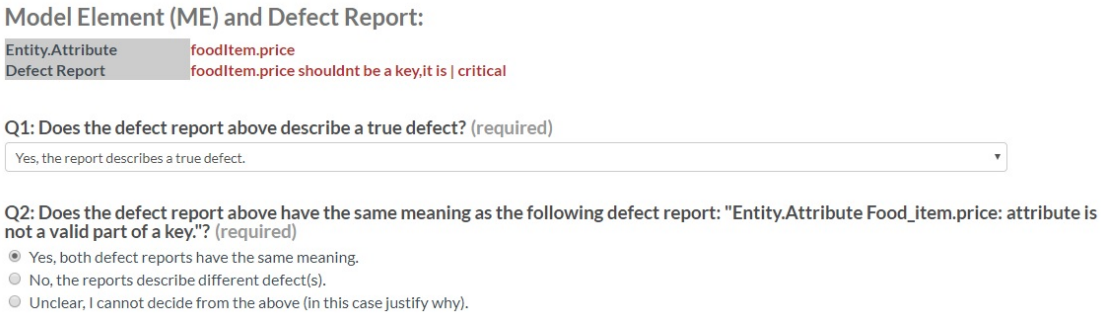


Figure 3.2: Guiding questions for the Defect Validation task.

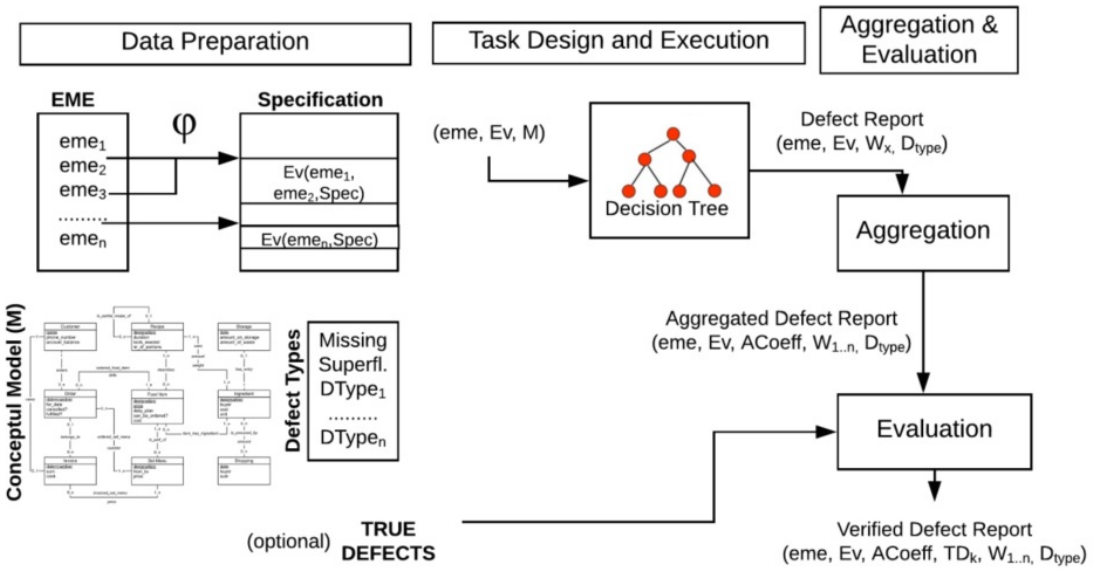


Figure 3.3: The VeriCoM Approach [23].

i.e., analyse whether the defect report indeed reported a defect and second compare the description with the description of the TrueDefect for the same ME.

3.2 The VeriCoM Approach

In the paper [23] a generic approach for Verifying Conceptual Models (VeriCoM) is proposed, fulfilling the need for a structured and formalized examination of Crowd-sourced Model Inspection (CSI) and verification with respect to a textual specification. Central element of VeriCoM is the Expected Model Element (EME), which enables the selection of core areas of a model, which are in need of inspection, thus reducing the amount of evaluated domain model elements and their relations for each individual human computation task. This enables the separation of the model and the textual

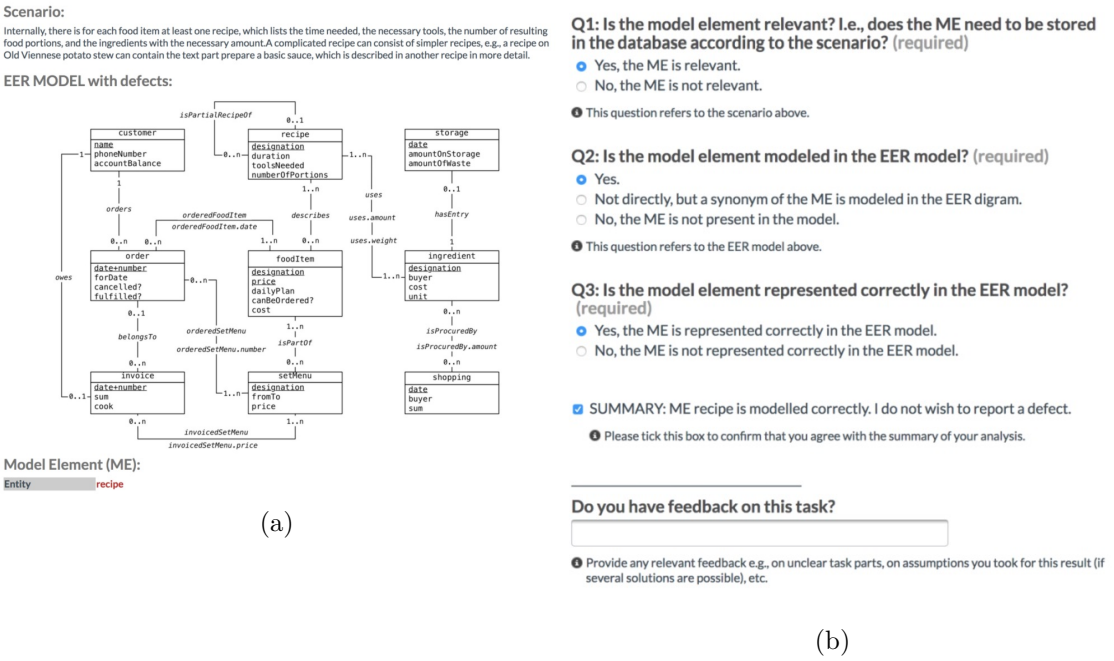


Figure 3.4: Model verification task containing (a) the model element, evidence scenario and model and (b) questions for verification guidance [23].

specification, into distinct scenarios encapsulating a specific set of EMEs. The focus of human workers can then be guided towards certain model areas, reducing the amount of cognitive workload for each worker. The main stages of this approach are depicted in Figure 3.3.

1. **Data Preparation:** The process starts with pairing the conceptualized model *M*, which should be verified, with it's corresponding textual specification *Spec*. The conceptual model hereby can be an entity relationship model ERM, a class diagram, a use case diagram, or any other kind of diagram, preferably specified in UML[1]. The specification is written in natural language and describes the desired functional and non-functional requirements and concepts of the model. Also the verification focus, that is, which model element(ME) types should be verified is defined. The focus can be laid on domain concepts ME_C and/or their relations ME_R.

    a) **Identification of EME:** The specification mentions elements, which are expected to appear in the conceptual model M, these need to be identified.

[1]http://www.uml.org/

This can be achieved by manually performed search, human computation and crowdsourcing or via natural language processing techniques, depending on the size of the specification. Combinations of those processes can also lead to the desired results. Ideally the specification is written as a structured user story, e.g. in the form of "As a ⟨*role*⟩ , I can ⟨*capability*⟩, so that ⟨receive benefit⟩.", which can potentially improve automated EME detection.

b) **Identification of evidence for the *EME* in the specification:** To map evidence EV for an eme to its occurrence in the specification *Spec*, an assignment function φ is required. φ should generate a set of evidences which suffice the properties of being (a) representative for each eme; (b) small enough to be applicable for human computation tasks and (c) capable of transporting the necessary context information to workers. Such a function can be implemented manually for smaller specifications, while again natural language processing techniques can deliver significant time saving benefits for larger and more complex specification texts. Automatic detection of relevant evidence(s) requires a decision on how to chose the evidence for a given eme. Included options are: (a) assigning the evidence where the eme is first mentioned; (b) choose the evidence (e.g., a paragraph) that most often mentions the eme; or (c) selecting all evidences which mention the eme, instead of a single representation.

c) **Definition of Defect Types:** Defects can be of different type. These include the two domain independent defect types *SUPERFLOUS* (i.e. EME's which are modelled, but not relevant) and *MISSING* (i.e. relevant EME's, which can not be found in the model). Besides these, a number of additional defect types should be defined in this step. Defect types ease the aggregation process and significantly improve evaluation speed. Task interfaces that guide workers towards identifying a specified defect type also reduce the amount of free-text defects, which need to be aggregated manually. An example for such a defect type would be *WRONG_KEY*, representing the defect of a wrongly chosen key attribute of an domain object, e.g. a non-unique name-attribute, which does not identify a customer. Defect Types which where defined for the CSI-Experiments are described in section 4.1.4.

2. **Task Design and Execution:** Detection of the different defect types defined in the previous step, can be guided through a thoughtful task design. This structured approach to detect defects of different type based on emes and their relevant evidence from the specification, can be tailored to the needs of the domain. The here proposed task design contains an underlying decision tree, depicted in figure 3.5, which guides the worker through the exercise, to identify a defect of a specific type, by asking questions about the eme and its appearance within the model.

a) **Relevance:** The first judgement asked from the worker, is whether the eme and its evidence are relevant in the context of the scenario and therefore should be present in the model or not. This question not only helps to identify defects

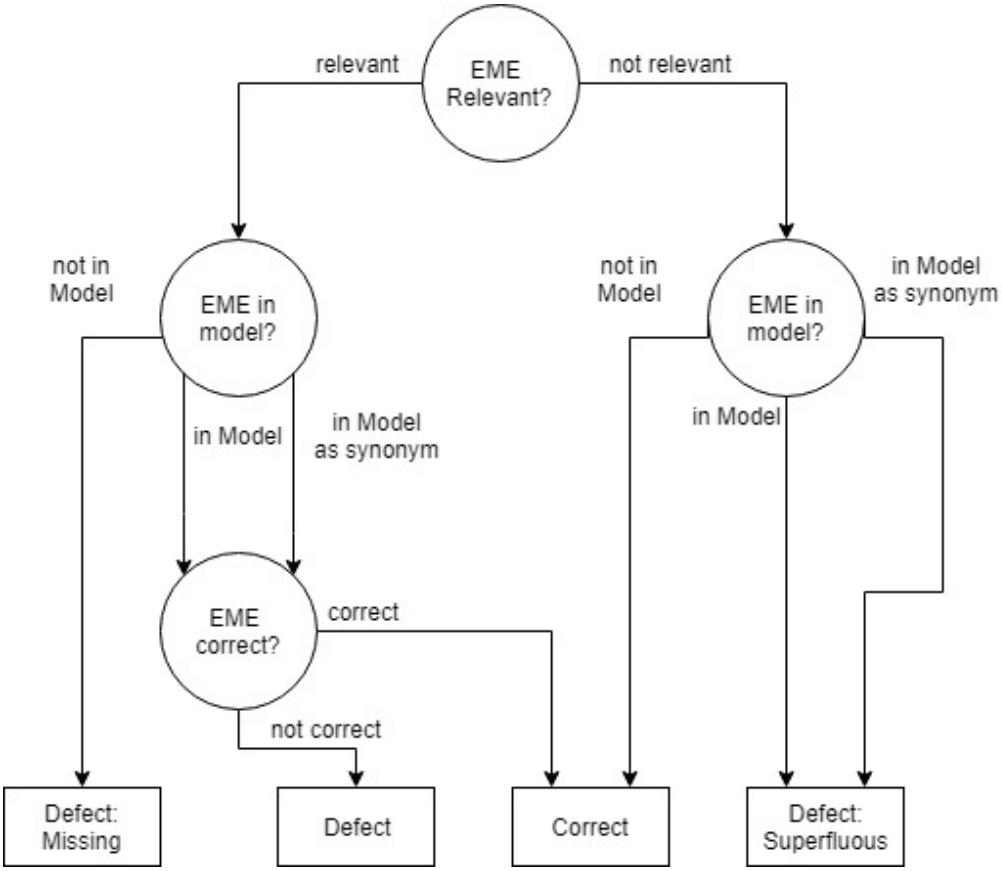


Figure 3.5: Decision tree underlying task design.

within the model, but can also shine light on the worker and its understanding of the scenario. Purposefully incorporated irrelevant emes with fake evidence can be used as control questions. Worker who continuously judge such emes wrong, can be given a lower weight in the overall judgement aggregation.

b) **Representation:** Next the workers attention is guided to the model itself and the location of the eme therein. The possible decision can be of one of three cases: (a) the eme or (b) a synonymous representation of the eme is present in the model; (c) the eme is missing from the model *M*. At this point, the two domain independent defect types superfluous and missing are already fully described. Identifying a non-relevant eme in the model states that a superfluous eme has been modelled. On the other hand, if a relevant eme can not be found in the model, a missing defect is reported. Non-relevant emes that are not in M lead to the conclusion that the model is correct, with regards to the eme in question. Locating a relevant eme, or a synonym thereof, in the model doesn't provided judgement about the correctness of the modelled element, which needs to be decided in the next question.

c) **Correctness and Interpretation:** In this part of the evaluation process, the worker is asked, whether the appearing relevant emes (or their synonyms) are correctly modelled in M. If incorrect modelling is detected, workers file a *defect report*, either as free text or an application domain specific typed defect. Each individual judgement for an eme and its evidence, is collected as individual Defect Reports (DR), which are of a specific Defect Type. This can be formalized as quadruples for each worker as $(W_x) : DR(eme, Ev(eme, Spec), W_x, D_{\text{type}})$.

3. **Aggregation:** Taking the defect reports for an eme by n workers as input, an aggregation can be performed to identify a final defect type, agreed upon by the majority of workers. For each defect type reported for that eme, we compute an agreement coefficient (*ACoeff*) as the inter-rater agreement on that defect type. The defect type with the highest *ACoeff* is considered the final defect type for the eme. If the opinion of the workers is tied between multiple defect types the defect is label as Undecided. This highlights the importance of defect types, as free-text defects have to be aggregated manually or performed within a secondary human computation task. The output of this aggregation step are *Aggregated Defect Reports* denoted as: $ADR(eme, Ev(eme, Spec), ACoeff, W_{1..n}, D_{\text{type}})$.

4. **Evaluation:** To check whether the defect detection process satisfies various performance metrics, an evaluation step is performed, to identify possible necessary improvements. In the case that previously defined *True Defects* (*TDs*) (i.e., known defects) are present, they could be matched with the collected defects *ADRs* and recall and precision metrics could be computed. In the absence of such gold standard defects, manual computation of the precision metrics can be performed, but no recall values can be computed. Due to the time intensity of the manual evaluation, a threshold value for the *ACoeff* should be considered, to reduce the number of *ADRs*. The output of this stage consists of verified defect reports, possibly aligned to a true defect $(TD_k) : VDR(eme, Ev(eme, Spec), ACoeff, TD_k, W_{1..n}, D_{\text{type}})$.

The VeriCoM approach is conceptualized in a generic way and is therefore applicable in a variety of model inspection use cases. In [23] the VeriCoM approach is applied to a Software Engineering (SE) use case, where, given a textual specification document, an EER model is checked for correctness and completeness. This SE use case forms the basis of the CSI-Experiment and the developed CSI-Platform described in this paper.

3.3 CSI-Experiment Stakeholders

Identification of the core users of the envisioned CSI-Platform, is key to understand their different needs as well as desired platform features and functionalities. Definition of requirements, which are beneficial for software developers, reference these stakeholders in their user-stories and with that, bind the requirement to a specific user group. For the CSI-Platform the identified user groups include:

- System-administrators: Responsible for system development and maintenance, this group consists of software engineering experts, who want to operate the system and provide a continuous service to the other stakeholders.
- Experiment-administrators: This group of users is composed of the scientific research staff and responsible for designing, running and evaluating the CSI-Experiments. Their key tasks are:
  - to define the experiment specification
  - to create the conceptual model
  - to identify model elements (if done manually)
  - to split specification into scenarios
  - to apply the evidence function (if done manually)
  - to define gold standard defects (i.e. TrueDefect)
  - to define domain specific defect types
  - to schedule and prepare workshops
  - to create tasks for each workshop
  - to execute the workshops
  - to trigger output aggregation and evaluation

  Experiment-administrators are the stakeholders benefiting the most from software tool support and addressing their needs is key for a successfully developed CSI-Platform.
- Workshop-administrators: Supporting the execution of a workshop, workshop-administrators gather participant information, brief and monitor the workers before and during the workshops and distribute them to the designated jobs in groups.
- Worker/Student/Participant: Workshop participants use the platform to get feedback on their performed tasks and to obtain information on the performance of their colleagues from the same workshop. This insight should improve their learning experience and enhance their Software Inspection and Model Analytics skills.

3.4 Requirement Analysis Method

To gather insight in the experiment process, briefing workshops have been held with the experiment administrators before testing the first prototype during the experiment run in autumn 2017. This section describes how information about the experiment process has been gathered.

Within the first meetings to elaborate the needs of the experiment team, the crowdsourced software inspection process was described and the experiment setup was analysed in detail, to get an overview of the current status as well as the vision and the desired outcome of the experiments. The goal of the CSI-Experiments is, to compare the novel VeriCoM-approach with the traditional pen and paper method. Therefore the experiment setup was designed to split up the group of participants into two groups, one performing the traditional pen and paper model inspection, to function as a control group and the other group testing the approach based on human computation and crowdsourcing. Following VeriCoM each performed step of the experiment was further analysed:

- Data Preparation: To enable the usage of crowdsourcing methods, a database schema was developed by the experiment team for the first experiments held in autumn of 2016. This database schema contained the basic entities and relations to describe the experiment setup, but was in need of enhancement to allow for the development of a supporting software tool.
- Task Design and Execution: The design of the data model could be traced to the usage of FigureEight (formerly CrowdFlower), as the database entities where loosely bound to the utilization of the services offered by the FigureEight platform. This boundedness to FigureEight, underlines the need for the definition of independent and generic data types, to allow connection to different crowdsourcing platforms and to enable the creation of a software prototype for the CSI-Experiments.
- Aggregation: Following the steps of VeriCoM, the data aggregation procedure was analysed next. This procedure is a purely manual step and is very time consuming due to the lack of a coherent type definition for the reported defects and the non-existent tool support. Each defect report was delivered in a free text format and had to be read and interpreted by a member of the experiment team. To overcome this issue, the free text defect reports of the earlier experiments served as input for the newly created defect validation task first introduced in the experiment of autumn 2017. Goal of the defect validation task was to align the free text defect reports with a predefined set of gold standard defects (also see Table 3.1).
- Evaluation: Examining the follow up evaluation step showed, that it was also done manually with the help of spread sheets. This step showed great potential for improvement through definition and introduction of common defect types for defect reports and automated evaluation (i.e. alignment of defect reports with gold standard defects) as well as computation of performance metrics. Again, no automated evaluation of the aggregated datasets was in place to reduce the workload of the experiment administrators.
- Feedback: During interviews with the experiment team, the desire to provide feedback for students on their performance during a workshop was stated. Giving individual feedback was not possible in the past, because computation of student performance metrics would have been a time consuming manual task. Providing

this individual feedback with a software tool would be very valuable for students and would enhance their learning outcome through participation in a CSI-Workshop.

With the VeriCoM process steps analysed and the major drawbacks identified, a proof of concept prototype of the CSI-Platform was developed and tested during the experiment run of autumn 2017. Goal of the test run was to get further insights into VeriCoM and to test applicability of the software automation support under live conditions.
At the beginning of the experiment, FigureEight jobs were defined and the datasets were assigned to the jobs. With the lack of a substitution for FigureEight, this first step was considered not to be in need of tool support in the first draft of the CSI-Platform. This is also true for task execution and distribution of tasks to the worker-crowd, as FigureEight services are adequate for the execution of the experiment jobs.
The focus of the prototype solution was on the most time consuming parts of VeriCoM, namely the interpretation of defect types and the aggregation and evaluation of defect reports. The developed prototype was able to identify the defect type base on the answers given by the workers and was capable of downloading and storing the data in a database for later analysis. Also a first version of student feedback was provided in the form of an individual email with performance metrics, including defect detection precision and recall.

Overall the proof of concept implementation showed promising results and participating in the management of the experiment, gave detailed insight in the required features and functionalities of a full fledged CIS-Platform. A detailed listing of the status quo of the CIS-Experiment process prior to autumn 2017 and the process steps selected for improvement, can be found in Section 3.5.

3.5 Status Quo of the CSI-Experiment process

Elaboration of the status quo of the CSI-Experiment process prior to this thesis, follows the VeriCoM approach and its consecutive steps. This section describes each of these steps and highlights major issues and drawbacks of VeriCoM without software tool support. These drawbacks (Dxx) are then translated into functional and non-functional requirements for the CSI-Platform in Section 3.6.

1. Data Preparation:
    - D11 - The data-model is only visualized in the form of database tables and excel spreadsheets.
    - D12 - Data can only be accessed through SQL-queries.
    - D13 - Preparation of datasets is managed with excel spreadsheets.
    - D14 - Selection of datasets for the experiments is cumbersome due to the lack of a user interface.

- D05 - Data model is FigureEight specific.

2. Task Design and Execution
   - D21 - Tasks have to be designed with a FigureEight specific domain language.
   - D22 - Distribution of participants to jobs requires the usage the TISS-platform [2], which is not integrated into FigureEight.
   - D23 - Execution of the crowdsourcing jobs and collectable judgements are limited in the free version of FigureEight.
   - D24 - The user interface of FigureEight is slow and not optimal for the CSI-Experiments.
   - D25 - No automatic download of workshop results.
   - D26 - Manual conversion of workshop results into data objects of the CSI-Experiment domain.
3. Aggregation
   - D31 - No automatic interpretation of defect types, only free text descriptions.
   - D32 - Free text descriptions can only be interpreted vaguely.
   - D33 - Aggregation of data is done manually, which makes it slow and time consuming.
   - D34 - Manually aggregating defect reports is inefficient and does not scale for larger participant groups.
   - D35 - Error-prone process, as data entries can be easily overlooked and distort the following statistical evaluation.
4. Evaluation
   - D41 - Manually matching defect reports with gold standard defects(i.e. determination of defect correctness) is inefficient and error-prone.
   - D42 - Computing performance metrics manually for different workshops requires many SQL-queries to the database.
   - D43 - Lack of data visualization.
   - D44 - Under-performing workers can not be filtered out.
5. Student-Feedback
   - D51 - No feedback for students participating in the workshops.
   - D52 - Low learning outcomes.
   - D53 - Workshop performance of students can not be reviewed.
   - D54 - No integration with existing e-learning platforms.

[2] https://tiss.tuwien.ac.at/

Table 3.2: VeriCoM 1.: Requirements for the Data Preparation step

1. Data Preparation:		
Requirement	**Drawback**	**User Story**
R11	D11	As an experiment administrator, I can access a visual representation (user interface, combined-tables, diagrams, etc.) of the data-model, so that I can easily see relationships between entities and get an overview of the experiment setup.
R12	D12, D13	As an experiment administrator, I can create, read, update and delete (CRUD) data entities through a user interface, so that I can prepare the experiment datasets with a uniform interface.
R13	D14	As an experiment administrator, I can combine data entries into dataset, so that I am able to select specific datasets for experiments and tasks.
R14	D15	As an experiment administrator, I want to have an independent data-model tailored to the CSI-Experiment use case and generic enough, so that i can use crowdsourcing-services other than FigureEight.

3.6 CSI-Platform Requirements

Analysing the drawbacks identified in Section 3.5 allows for the elicitation of requirements for the envision CSI-Experiment support platform. Comprehensive system requirements are the basis for a software development project, which addresses functional and non-functional system specifications and incorporates the needs of key stakeholders. These requirements are translated into user-stories in the for of "As a ⟨role⟩, I can ⟨capability⟩, so that I ⟨receive benefit⟩.". The defined user-stories can then be taken up by a software developer to produce the desired feature that fulfils the stakeholders needs.

Based on the drawbacks (Dxx) identified in Section 3.5, requirements (Rxx) have been derived and translated into a user-story. The requirements stated in the Tables 3.2,3.3, 3.4, 3.5 and 3.6, display the requirement elicitation for each step of VeriCoM (except 5. student feedback), together with the drawback they have been derived from.
Only the most important requirements have been defined and the requirements in the tables do not represent a complete list of the desired features for the CSI-Platform. Within the software prototype the user-story candidates which improve the CSI-Experiment process the most (i.e. reduce manual work, improve time consumption etc.), have been included and implemented in the platform. Therefore only a selection of the defined requirements have been directly implemented as the platform represents an early prototype and some requirements have to be addressed in future work (see Section 7.3).

Within this chapter the CSI-Experiment process was analysed and its application in the

Table 3.3: VeriCoM 2.: Requirements for the Task Design and Execution step

2. Task Design and Execution:		
Requirement	**Drawback**	**User Story**
R21	D21	As an experiment administrator, I want to be able to design tasks independent the FigureEight specific domain language, so that I can tailor the tasks to better fit the CSI-Experiments.
R22	D22	As an experiment administrator, I want to integrate an external (existing) platform into the experiment process, so that management of participant is easier.
R23	D23	As an experiment administrator, I can execute many jobs and collect many judgements, to increase the amount of data for the evaluation step.
R24	D24	As an experiment administrator, I can work with a fast reacting and responsive user interface.
R25	D25	As an experiment administrator, I can download and export workshop results in the form of .csv-files.
R26	D26	As an experiment administrator, I am able to import .csv-files and the platform automatically created corresponding data object, so that I can easily transfer and process (legacy) data.

Table 3.4: VeriCoM 3.: Requirements for the Aggregation step

3. Aggregation:		
Requirement	**Drawback**	**User Story**
R31	D31, D32	As an experiment administrator, I receive an automatic interpretation of worker answers into defect types, so that manual interpretation of free text descriptions is reduced to a minimum.
R32	D33	As an experiment administrator, I can access automatically aggregated data with predefined aggregation options, so that manual aggregation becomes obsolete.
R33	D34	As an experiment administrator, I want automatic aggregation to be scalable to large sets of data, so that I can improve the quality of evaluation results through working with large groups of participants.
R34	D35	As an experiment administrator, I want to have a thoroughly tested aggregation algorithm, so that I can be sure that the statistical evaluation is correct.

Table 3.5: VeriCoM 4.: Requirements for the Evaluation step

4. Experiment result evaluation:		
Requirement	**Drawback**	**User Story**
R41	D41	As an experiment administrator, I want to be able to automatically match a selected set of defect reports (per workshop(s) / worker) with gold standard defects, so that I can determine if a defect report is correct and so that I can evaluate different sets of data.
R42	D42	As an experiment administrator, I want to automatically compute performance metrics for different selected sets of data (without manual SQL-queries to the database), so that I can compare different workshops with each other.
R43	D43	As an experiment administrator, I want to have a visual representation of the workshop performance metrics, so that I can easily see all necessary metrics on one dashboard.
R44	D44	As an experiment administrator, I want to filter out underperforming workers of the evaluation, so that I can reduce distortion of the statistical evaluation.

Table 3.6: 5.: Requirements for the Student-Feedback step

5. Student-Feedback:		
Requirement	**Drawback**	**User Story**
R51	D51	As an experiment-/workshop-administrator, I want to provide feedback (e.g. in the form of e-mail) to the students participation in the workshops, to enhance their learning experience.
R52	D52, D53	As an experiment-/workshop-administrator, I want to be able to see the performance (i.e. number of correctly identified defects, detection precision, recall, group agreement etc.) of all students of an experiment run, so that I can improve the task design/tutorials/teaching, if necessary.
R53	D53	As a workshop participant/student, I want have a graphical overview of my performance during a workshop, so that I can see how I performed and where I still need to improve my skills.
R54	D54	As a experiment-/workshop-administrator/ workshop participant/student, I want to have an integration of the new support platform into existing (commonly used) e-learning platform, so that I can access the platform from a well known site.

form of VeriCoM was described. Through participation in introduction workshops and the CSI-Experiment of autumn 2017 and the development of a proof of concept prototype, the stakeholders of the experiments were identified together with the difficulties and drawbacks they have to face while performing the experiments without software tool support. Based upon these drawback requirements have been defined which should be addressed by the CSI-Platform. The identified requirements offer an answer to research question RQ2. The requirements elicitation shows shortcomings in the data model and the need for automation in the aggregation and evaluation phases of VeriCoM, as well as the wish for automated student feedback. In Chapter 4 the identified requirements are used to defined and enhance the data model and to formalize algorithms to automate VeriCoM process steps.

CHAPTER 4

Data Model Design and Algorithm Development

This chapter summarises the design-phase of the software development life-cycle (SDLC [20]) for the CSI-Platform. It takes into account the requirements for VeriCoM, identified in the previous Chapter 3 and consists of a detailed description of the most important data model elements (Section 4.1), the definition and formalization of developed algorithms, the result-converter and interpreter (Section 4.2.1), as well as the components for defect report aggregation (Section 4.2.2), experiment evaluation (Section 4.2.3) and student feedback-provision (Section 4.2.4).

4.1 Data Model

Following VeriCoM, the designed data models incorporate the formalizations made in paper [23] and expand them with additional types needed for software development and also modelling their relations to each other.
The core elements of VeriCoM are described in the following sub-sections. Figure 4.1 shows the Extended-Entity-Relationship(EER)-diagram of the data model for the VeriCoM-approach. To explain the concepts and their applications in the data model, a running example is provided for the formalized elements.

4.1.1 M - Domain Model, and Spec - Specification

A conceptual domain model (M) consists of a collection of model elements (me) of different types. The complete set of model elements (ME) contains all the elements the conceptual domain model is comprised of, i.e. $M = ME = \bigcup_n ME_1 \dots ME_n$, where $ME_1 \dots ME_n$ describe the sets of model elements of different types, such that

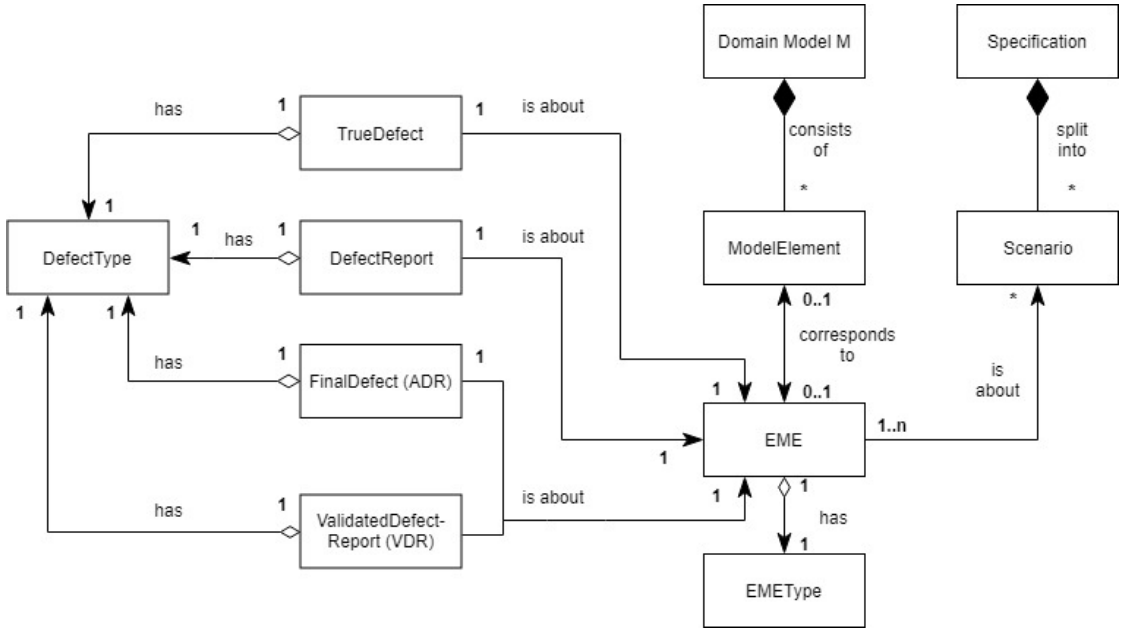


Figure 4.1: VeriCoM approach data model.

$M = \bigcup_n ME_n$. $M = ME_C \bigcup ME_R$, describes the minimal union of at least the two model element set ME_C, the model element concepts in a domain, and ME_R, their relations.

The verification of a conceptual domain model (M), with respect to a frame of reference (FR) is formally defined as a function (γ), which, if applied to the model (M) and the frame of reference (FR) leads to the identification of defects (D).

$$\gamma(M, FR) \rightarrow D \tag{4.1}$$

A *Frame of Reference* (FR) is a knowledge source complementary to the domain model M, e.g. a system specification that describes (approximately) the same domain knowledge as the model. The focus of the CSI-Experiment lies on the verification of models guided by one form of a FR, a textual specification (*Spec*).

4.1.2 EME - Expected Model Element

Expected Model Elements (*emes*) build the core concept of the crowd-sourced model inspection and function as a reference point to the EER model (i.e., graphical representation of M), the given answers of the workers during the human computation task and the types derived from the task results. They describe a certain model element (e.g. entities, attributes, relationships and relationship attributes), which is mentioned in the textual specification *Spec* and is expected to be found during model inspection in the

EER model diagram. A representative evidence (EV) from *Spec* has to be assigned to each eme (apart from superfluous emes), which is formally a function φ defined as:

$$\varphi(EME, Spec) \rightarrow EV_{EME,Spec} \tag{4.2}$$

The set EME overlaps, but must not be identical with the set of model elements in M. The intersection of the expected and the actually modelled model elements contains all those emes for which an equivalent model element me exists (i.e., the same as or a synonym of eme denoted with $\approx$) [23].

$$EME \cap ME = \{eme | \exists me \in ME \wedge eme \approx me\} \tag{4.3}$$

An example for the inequality of the two set EME and ME would be superfluous emes, i.e. elements which are not relevant according to the specification but nevertheless modelled. These emes can be defined as a measure of checking the experience level of workers.
References to emes are used by other classes of the data model to be able to aggregate and match different objects, e.g. a defect reported by a worker for an eme and the corresponding TrueDefect (i.e. gold standard defect) for the same eme. A detailed description of aggregation and matching with TrueDefects (i.e. evaluation of correctness) can be found in Sections 4.2.2 and 4.2.3.

EMEType

Each eme is of a certain *EMEType*, which specifies the building block of the model diagram the eme describes (e.g, entity, attribute, etc.). The following EMETypes have been defined for an EER diagram:

$$\begin{aligned} EMETypes = \{&ENTITY, ENTITY\_ATTRIBUTE, RELATIONSHIP, \\ &RELATIONSHIP\_ATTRIBUTE, RELATIONSHIP\_MULTIPLICITY\} \end{aligned} \tag{4.4}$$

The defined EMETypes correspond to the following elements of a model:

- `ENTITY` - An entity mentioned in the specification, e.g. *customer*.
- `ENTITY_ATTRIBUTE`, - An attribute-field of an entity, e.g. *customer.accountBalance*.
- `RELATIONSHIP` - A relationship between two entities, e.g. *(customer + has + account)*.
- `RELATIONSHIP_ATTRIBUTE` - An attribute of an entity relation, e.g. *(shoppingList + contains + ingredient).price*.
- `RELATIONSHIP_MULTIPLICITY` - The multiplicities of the relationship e.g. *(foodItem(1..n) + isPartOf + setMenu(0..n))*.

An advantage of typed emes is, that they can be used to further guide the worker through model inspection, as emes of certain type, can have a specific set of predefined defects, which the worker can choose from. For example, when a worker has to evaluate an eme of type *RELATIONSHIP_MULTIPLICITY*, the task can be designed to give "Wrong relationship multiplicity." as a predefined choice for reporting a defect. Also, EMETypes further clarify the *DefectType* (see section 4.1.4) of a reported defect. Table 4.1 shows examples of emes with their corresponding type.

Table 4.1: EME example

EME ID	**Text**	**EMEType**
EME05	order	ENTITY
EME22	order.advancePaymentAmountReceived	ENTITY_ATTRIBUTE
EME23	order.canceled?	ENTITY_ATTRIBUTE
EME24	order.dateAndNumber	ENTITY_ATTRIBUTE

4.1.3 Sc - Scenario

The textual specification *Spec* for a conceptual domain model M can be split up into different parts, called *Scenarios* in the VeriCoM context. These scenarios can be comprised of single sentences or text paragraphs of the original specification and are defined manually by the experiment administration team. A scenario *sc* should describe a part of the model and its contained model elements in detail, i.e. there should be evidence for certain model elements expected to be present in the conceptual domain model. Let $Ev_{eme,Spec}$ be a relevant evidence for an eme within a specification *Spec* as an arbitrary long text chunk from *Spec* where the eme is mentioned [23], then a scenario consists of the textual concatenation (represented by $\sum$) of one or multiple evidences for emes. A Scenario *sc* can therefore be formalized as:

$$sc = \sum_{1}^{n} Ev_{eme_n,Spec}. \tag{4.5}$$

The function φ (4.2) can also be formalized more specifically in the context of a scenario. Let φ_{sc} be the function assigning each eme to the scenario where it is mentioned.

$$\varphi_{sc}(EME, Sc) \rightarrow EV_{EME,Sc} \tag{4.6}$$

While reading the scenario description the focus of a worker is already guided to a particular area of the model, which eases detection of defects for the emes in this area. Evidence for an eme within a scenario, is modelled as a reference table, respectively assigning the emes to those scenarios, for which evidence was found in their scenario-description.

The set of scenarios (*Sc*) has to completely describe the whole textual specification (*Spec*) and additionally offer descriptive text to guide the worker during model inspection.

$$Sc = \left\{ sc | sc \subset Spec \wedge \bigcup sc = Spec \right\}. \tag{4.7}$$

However, *Sc* has to minimally include all the evidences EV for all the emes mentioned in Spec ($EV_{EME,Sc} = EV_{EME,Spec}$). Table 4.2 shows an example scenario and some evidences for corresponding emes within this scenario.

Table 4.2: Scenario with evidences for emes example.

Type	**ID**	**Text**
Scenario	Sc1	During an order, the customer composes for his guests a selection of set menus or individual food items listed in the menu. During the order the customer has to declare when the meal should take place and whether the meal will be eaten at the restaurant or will be taken out. For any order beyond 150.- Euro, an advance payment of around 10% has to be provided. For each order taken, the customer receives an order number, which he can use to cancel the order. An advance payment expires, if the related order is cancelled.
EV(order)	EV(EME05,Sc1)	"During an order,..."
EV(order.advancePayment-AmountReceived)	EV(EME22,Sc1)	"For any order beyond 150.- Euro, an advance payment of around 10% has to be provided."
EV(order.canceled?)	EV(EME23,Sc1)	"An advance payment expires, if the related order is cancelled."
EV(order.dateAndNumber)	EV(EME24,Sc1)	"..,the customer receives an order number„..."

4.1.4 TD - TrueDefect

To evaluate the *VeriCoM* approach within a controlled experiment by a crowd of workers, the conceptual model has to be seeded with a predefine set of defects, purposefully modelled to be found by the workers, to test their skills and also to provide means of

measuring and evaluating their inspection performance. Therefore a set of *TrueDefects* (*TD*) has been defined for defective elements in the model. These *TD* are triples connecting an eme with its evidence and assigning a predefined defect type (D_{type}).

$$TD\,(eme, Ev(eme, Spec), D_{type})\,. \tag{4.8}$$

The defect type is assigned based on how the corresponding eme was erroneously modelled and the type of the eme. An example for a TrueDefect is given in Table 4.3.

DefectType

A *DefectType* further specifies the kind of defect that is being reported for an eme. TrueDefects, DefectReports (*DR*) (described in Section 4.1.5) as well as AggregatedDefectReports (*ADR*) (see Section 4.1.6) have a *DefectType*.
Grouping defects by type allows for (a) evaluation of correctness of a DefectReport by trying to match it with a TrueDefect and (b) aggregation of various DefectReports, stated for the same eme by different workers, into *AggregatedDefectReports*. Formally the set of DefectTypes is defined as:

$$\begin{aligned} DefectTypes = \{&MISSING, SUPERFLUOUSE, SUPERFLUOUS\_EME,\\ &SUPERFLUOUS\_SYN, WRONG, WRONG\_SYN, WRONG\_RELM,\\ &WRONG\_RELM\_SYN, NO\_DEFECT, UNDECIDABLE\} \end{aligned} \tag{4.9}$$

While the *DefectTypes* `MISSING` and `SUPERFLUOUS`, assigned to elements of the sets $ME \setminus EME$ and $EME \setminus ME$ respectively, are domain independent, other defect types defined here are specific to EER-diagrams. The defect types present in the conceptual model of VeriCoM include:

- `MISSING` - model element is mentioned in the specification and relevant but not present in the conceptual model.
- `SUPERFLUOUS`, `SUPERFLUOUS_EME`, `SUPERFLUOUS_SYN` - element is not relevant for the model but depicted as an eme or a synonymous representation thereof.
- `WRONG`, `WRONG_SYN`, - describes a defect which is not categorized and in need of further textual description by the worker.
- `WRONG_KEY`, `WRONG_KEY_SYN` - defect for an eme (or a synonymous representation) of type `ENTITY_ATTRIBUTE`, which has been wrongly chosen to be a key attribute.
- `WRONG_RELM`, `WRONG_RELM_SYN` - defect for an eme (or a synonymous representation) of type `RELATIONSHIP_MULTIPLICITY`, for which the multiplicities have been modelled incorrectly.

- `NO_DEFECT` - a *DefectReport* which states no defect, i.e. a correctly modelled element.

- `UNDECIDABLE` - this type is exclusively for *AggregatedDefectReports* and states that there was no agreement between workers on a defect type during the aggregation process.

Table 4.3: TrueDefect example.

TD	**EME**	**Ev(eme, Spec)**	D_{type}	**Description**
D21	EME22	EV(EME22, Sc1)	MISSING	order.advancePaymentAmountReceived: attribute is MISSING.

TDN - TrueDefectNeighbourhood

During the first test-run of the CSI-Experiment, workers correctly reported defects on related emes, rather than the eme in question. This led to the notion of a neighbourhood around a TrueDefect (*TD*), referring to a defect of an eme with a similar name or within a close context to the original eme which was asked to be evaluated. Reporting a defect for an eme within its *TrueDefectNeighbourhood* (*TDN*), should therefore also be counted as a correctly stated defect report. This TrueDefectNeighbourhood is defined manually by the experiment task designers, as knowledge of the specification scenarios and its entity relations is required. The function μ assigns an eme with a corresponding TrueDefect td to its TrueDefectNeighbourhood.

$$\mu(eme, td) \rightarrow TDN_{eme,td} \tag{4.10}$$

TrueDefectNeighbourhood come with a *"relevance"* value attached, describing their proximity to the original eme.
An example for such a neighbourhood relation would be the entity attribute "customer.name" and its corresponding entity "customer". Defects reported for the eme "customer" already mentioning a defect for the true defect "customer.name" should also be counted as correct. Table 4.4 shows the TrueDefectNeighbourhood for the eme "customer".

Table 4.4: TrueDefectNeighbourhood example.

TDN	**TDN(EME)**	**TD**	**TD(EME)**
TDN1	EME01 - customer	D11	EME12 - customer.name
TDN2	EME01 - customer	D12	EME11 - customer.contactAddress
TDN3	EME01 - customer	D13 (Superfluous)	no corresponding eme - customer.phoneNumber

4.1.5 DR - DefectReport

The set of DefectReports (*DR*) is an interpreted version of the answers of a worker during the human computation task of an experiment workshop. A *DR* is formalized as a quadruple, connecting an eme and its evidence to a defect type, based on the judgements of a worker (W_x):

$$DR\,(eme, Ev(eme, Spec), W_x, D_{type})\,[23]. \tag{4.11}$$

As mentioned, a DefectReport is of a specific *DefectType* (see Section 4.1.4), with the type *"NO_DEFECT"*, stating that the eme in question was judged to be correctly modelled. Defect reports of this type can therefore be filtered out during the aggregation step of VeriCoM.
Two example defect reports can be seen in Table 4.5, where the first table entry (DR01) describes a missing eme, while the second entry (DR02) states a correctly modelled entity.

Table 4.5: DefectReport example.

ID	**EME**	**Ev(eme, Spec)**	**Worker** W_x	D_{type}
DR01	EME22 - order. advancedPayment-AmountReceived	Ev(EME22, Sc1)	W_1	MISSING
DR02	EME23 - order. cancelled?	EV(EME23, Sc1)	W_2	NO_DEFECT

When the defect type *"WRONG"* is derived from the experiment results, an additional textual description of the defect is required and manual evaluation, either through the experiment administrators or a follow-up human computation task, is necessary, to (a) determine whether the description matches a TrueDefect, (b) a new defect, unanticipated during model conceptualization, has been found or (c) a more specific defect type can be assigned to the DefectReport.

4.1.6 ADR - AggregatedDefectReport / FDR - FinalDefect

Multiple DefectReports for the same eme, reported by n workers (W), can be aggregated to an *AggregatedDefectReport* (*ADR*), defined as:

$$ADR(eme, Ev(eme, Spec), ACoeff, W_{1..n}, D_{\text{type}})[23] \tag{4.12}$$

in the VeriCoM approach in Section 3.2.

For each *AggregatedDefectReport ADR* the defect type of that report D_{type}, is aggregated and an agreement coefficient *ACoeff* is computed as the inter-rater agreement on that defect type. This *ACoeff* determines the final defect type and can be subject to

a optional filter, which only considers the defect types where the worker's agreement was above a certain threshold value. If there was a tie between two or multiple defect types (or optionally the threshold was not surpassed), then the designated defect type *UNDECIDEABLE* is assigned to the *AggregatedDefectReport*. A detailed definition of the the defect type aggregation and the computation of *ACoeff*, is described in Section 4.2.2 and Algorithm 4.2. Table 4.6 shows an ADR example together with the defect reports that were taken as input for the aggregation function.

Table 4.6: AggregatedDefectReport example.

DefectReports for EME22					
ID	**EME**	**Ev(eme, Spec)**	**Worker** W_x	D_{type}	
DR01	EME22	Ev(EME22, Sc1)	W_1	MISSING	
DR02	EME22	Ev(EME22, Sc1)	W_2	MISSING	
DR03	EME22	Ev(EME22, Sc1)	W_3	NO_DEFECT	
DR04	EME22	Ev(EME22, Sc1)	W_4	SUPERFLUOUS	
AggregatedDefectReport for EME22					
ID	**EME**	**Ev(eme, Spec)**	**ACoeff**	**Worker** W_x	D_{type}
ADR01	EME22	EV(EME22, Sc1)	0.5	$W_{1...4}$	MISSING

Defect reports given as a free-text description or of type *WRONG* with additional free-text description, have to be aggregated manually or by a follow-up human computation task (see DefectValidation in Section 3.1.1).

Note that in the following sections an AggregatedDefectReport (ADR), especially in Chapter 5, is often referred to as a *FinalDefect* (*FDR*) which should be seen as a synonymous representation of the same object ($ADR = FDR$).

4.1.7 VDR - VerifiedDefectReport

VerifiedDefectReports (*VDR*) mark the output of the final Evaluation-step of the VeriCoM approach. Within the Evaluation-step, ADRs are matched with the set of predefined TrueDefects. A VDR is therefore formally defined as:

$$VDR(eme, Ev(eme, Spec), ACoeff, TD_k, W_{1..n}, D_{\text{type}})[23]. \tag{4.13}$$

Table 4.7 depicts such a match between TD D21 and ADR01 for the eme EME22.

Table 4.7: AggregatedDefectReport example.

TrueDefect for EME22				
TD	**EME**	**Ev(eme, Spec)**	D_{type}	**Description**
D21	**EME22**	EV(EME22, Sc1)	MISSING	order.advancePaymentAmountReceived: attribute is MISSING.

Aggregated-/VerifiedDefectReport for EME22						
ID	**EME**	**Ev(eme, Spec)**	**ACoeff**	TD_k	**Worker** W_x	D_{type}
ADR01	**EME22**	EV(EME22, Sc1)	0.5		$W_{1...4}$	MISSING
VDR01	**EME22**	EV(EME22, Sc1)	0.5	**D21**	$W_{1...4}$	MISSING

4.1.8 Enums

In order to represent the answers to the guiding questions (see Figure 3.4) in a coherent way, which is independent of the platform used for crowdsourcing, enumeration types(enums) had to be defined. These enums are used to represent worker judgements on fields which have more than one possible answer. This eases the interpretation of incoming results of FigureEight, as theses enum values have been incorporated in the designed jobs.

ModelAnalysis Answer-Types

The answer values for FigureEight jobs of type *ModelAnalysis* have been defined for two reasons. First, to represent the three possibilities of the IsModelled-question, with its additional value *"SYNONYM"* and second, to help workers with their judgement on the defect type, which they want to report, by giving them a set of choices based upon common defects for the type of eme they got presented. For example if they should judge a relationship between two entities, a common error would be incorrect multiplicities. If that error is the case, they can directly report it, by choosing this answer without the need to add explicit description. This reduces the amount of free-text defects that are reported and improves later aggregation. If the answer *"DEFECT_OTHER"* is chosen, they can report a free-text report, in case the predefined possibilities do not reflect the error in the model.

The following answer-type sets have been defined:

$$IsModelledAnswer = \{TRUE, FALSE, SYNONYM\} \tag{4.14}$$

$$\begin{aligned} DefectJudgementAnswers = \{&DEFECT\_NOT\_A\_VALID\_KEY,\\ &DEFECT\_INCORRECT\_MULTIPLICITY,\\ &DEFECT\_OTHER, DEFECT\_NOT\_A\_VALID\_KEY\_SYN,\\ &DEFECT\_INCORRECT\_MULTIPLICITY\_SYN, DEFECT\_OTHER\_SYN\} \end{aligned} \tag{4.15}$$

DefectValidation Answer-Types

For the *DefectValidation* jobs the answer-types represent the answers of the workers to the task questions. The following answer-types have been defined:

$$IsTrueDefectAnswer = \{YES\_TRUE\_DEFECT, NOT\_TRUE\_DEFECT, NO\_INCOMPLETE, CANNOT\_DECIDE\} \tag{4.16}$$

$$DefectMeaningAnswer = \{YES\_SAME\_MEANING, NO\_DESCRIBES\_DIFFERENT\_DEFECTS, UNCLEAR\} \tag{4.17}$$

In this section the data model for VeriCoM has been defined. All necessary entities to verify a conceptual domain model M with respect to a textual specification *Spec* and the usage of expected model elements *EMEs* have been formalize. Also defect types for DefectReports, TrueDefects and the newly introduced concept of the TrueDefectNeighbourhood have been described and their relations to each other have been depicted. These entities and relations are detailed in the form of a running-example, which showcases some real example values for each of the introduced entities. Furthermore enum-types for the answers to the tasks ModelAnalysis and DefectValidation are defined. This data model answers the first part of research question RQ2 and forms the basis for VeriCoM as well as the developed algorithms to automate the VeriCoM process steps.

4.2 Algorithms

This section describes the developed algorithms for the CSI-Platform in detail. These algorithms build upon the data models defined in Section 4.1 and are used to convert and interpret results from *FigureEight* [1] into domain objects (see Section 4.2.1), to aggregate the DefectType of DefectReports (see Section 4.2.2), to evaluate CSI-Experiments (see Section 4.2.3) and to provide feedback to students (see Section 4.2.4).

4.2.1 Result-Conversion and Interpretation Algorithms

The developed CSI-Platform uses the services of *FigureEight* for the management of the crowdsourcing tasks. Human computation jobs are created and distributed to the workshop participants by *FigureEight* and the results are automatically downloaded as .csv-reports. These reports contain the answers of the participants to all the questions they have been asked during their human computation tasks in a raw tabular format. Therefore components for conversion and interpretation into CSI-Platform domain objects have been developed. Figure 4.2 depicts the designed entities and their relations.

[1] https://make.figure-eight.com/

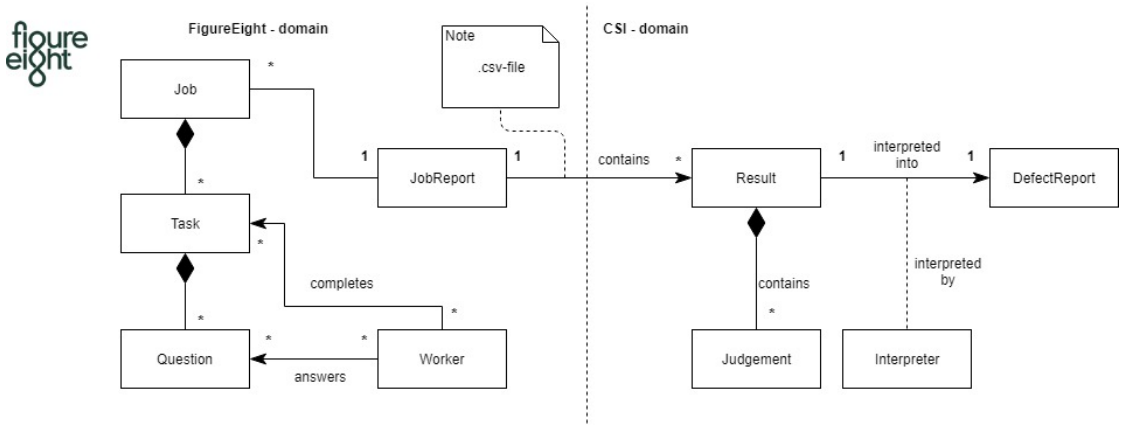


Figure 4.2: Interpretation/Conversion of FigureEight job reports into defect reports.

To determine the defect type of the defect report created during the conversion process, the Algorithm 4.1 was defined.This algorithm mirrors the decision tree defined in Figure 3.5 and differentiates additionally defined defect types.
First the output type is set to *NO_DEFECT*, referencing the state when a worker does not want to report a defect for the given model element. The tree of if-statements starts in the order of the guiding questions with the check if a relevant eme was modelled and consecutively modelled correctly (1-4). If the element is incorrectly represented, the defect judgement of the worker is analysed. This defect judgement can be one of 3 distinct enum values (see Section 4.1.8, not taking into account synonym values), which correspond to the type of the eme that was in question. In each case statement the defect type that reflects the answer is assigned to the return variable type (5-15). Likewise, the defect types for a synonymous representation of the modelled element are assigned in the second "else-if"-branch (16-29). If a relevant model element is absent, the defect type *"MISSING"* is returned (30). In the last branch the types for superfluous elements are assigned (33-36) and the resulting type is returned (37). Note that the defect types for synonyms are differentiated, but are not distinctly counted in the later evaluation of defects reported within a workshop. Instead they count to their original defect type, e.g. *DefectType.WRONG_KEY_SYN* counts to *DefectType.WRONG_KEY* and so forth. For the further aggregation, evaluation and feedback generation steps, only DefectReports are consulted.

4.2.2 Aggregation - Defect type aggregation

Multiple *DefectReports* for the same eme, are aggregated into a single *AggregatedDefectReport (ADR)* with a specific DefectType, that is defined through aggregation of the types of the initial defect reports (see Figure 4.3).

Algorithm 4.1: Interpretation of ModelAnalysisJudgements into DefectTypes

Input: A ModelAnalysisJudgement judgement
Output: A DefectType type

```
1  type ← DefectType.NO_DEFECT;
2  if judgement.isRelevant == true then
3      if judgement.isModelled == true then
4          if judgement.isCorrect == false then
5              switch judgement.defectJudgement do
6                  case DEFECT_NOT_A_VALID_KEY
7                      type ← DefectType.WRONG_KEY;
8                  end
9                  case DEFECT_INCORRECT_MULTIPLICITY
10                     type ← DefectType.WRONG_RELM;
11                 end
12                 case DEFECT_OTHER
13                     type ← DefectType.WRONG;
14                 end
15             endsw
16     else if judgement.isModelled == synonym then
17         if judgement.isCorrect == false then
18             switch judgement.synDefectJudgement do
19                 case DEFECT_NOT_A_VALID_KEY_SYN
20                     type ← DefectType.WRONG_KEY_SYN;
21                 end
22                 case DEFECT_INCORRECT_MULTIPLICITY_SYN
23                     type ← DefectType.WRONG_RELM_SYN;
24                 end
25                 case DEFECT_OTHER_SYN
26                     type ← DefectType.WRONG_SYN;
27                 end
28             endsw
29     else
30         type ← DefectType.MISSING;
31     end
32 else
33     if judgement.isModelled == true then
34         type ← DefectType.SUPERFLUOUS_EME;
35     else if judgement.isModelled == synonym then
36         type ← DefectType.SUPERFLUOUS_SYN;
37 return type;
```

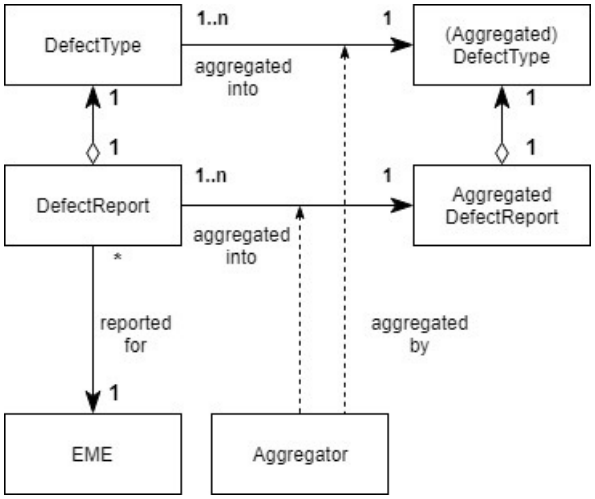


Figure 4.3: Aggregation data model.

The type of the ADR is the type, which the majority of the experiment participants agreed upon, i.e. reported a defect, for a certain eme with the specific type. Let ψ be the function to determine the aggregation of the defect types $DT_{1...n}$ for the EME eme_i into a single aggregated defect type T_{eme_i}, formalized as:

$$\psi_{eme_i}(DT_{eme_i,1...n}) \rightarrow T_{eme_i}. \tag{4.18}$$

Algorithm 4.2 showcases the implementation of the defect type aggregation function ψ. The algorithm takes the array of DefectReports, for which the defect type should be aggregated, as input, alongside the array of all the DefectTypes defined for the applied domain (see section 4.1.4) and an occurrence-Map, with the DefectTypes as key and an integer value as an occurence-counter. As output-parameter a DefectType T is returned. In the first part of the algorithm (row 1-7), the occurrences of each DefectType t within the DefectReport-array are counted. Then the maximum value of the occurrences-Map *maxOccurences* is determined (8) and its corresponding key is assign to T (9). Now the entry with the maximum value is removed from the map and it is checked if another entry with the same value is present in the map after the removal. If that is the case, i.e. workers could not decide between two or more DefectTypes, then the returned type T is set to *DefectType.UNDECIDEABLE* (10-14).

Determination of defect type occurrences of the above mentioned algorithm can be encapsulated into a separate utility function. This can be useful, as the computation of the agreement coefficient($ACoeff$) for the defect type T can then use the *occurrences*-Map, precisely its maximum value *maxOccurrence*, and the number of DefectReports n as input parameters. The $ACoeff$ is defined as:

$$ACoeff_T = \frac{maxOccurrence_T}{n}. \tag{4.19}$$

Algorithm 4.2: DefectType aggregation

Input: An array of DefectReports DR for an eme, An array of DefectTypes DT, A Map<DefectType, int> occurences
Output: An aggregated DefectType T

```
1  for each t ∈ DT do
2  |  for each dr ∈ DR do
3  |  |  if dr.type == t.type then
4  |  |  |  occurences(t) + +;
5  |  |  end
6  |  end
7  end
8  maxOccurence ← occurences.max();
9  T ← occurences.getKey(maxOccurence);
10 occurences.removeEntry(maxOccurence);
11 if occurences.max() == maxOccurence then
12 |  T ← DefectType.UNDECIDEABLE;
13 end
14 return T;
```

The $ACoeff$ is a value between 0 and 1, which measures the ratio of the defect report type mentioned most often for a given eme and all reported defects for that eme.

4.2.3 Experiment Evaluation Components

For the evaluation step of the VeriCoM approach, i.e. the check whether a given aggregated defect report *ADR* also corresponds to a predefined TrueDefect, a component was designed, that aligns the given reports with the respective TrueDefect and created a *VerifiedDefectReport VDR* for each ADR that could be matched (see Figure 4.4). The resulting sets of matched and unmatched defect reports, is then used to evaluate the performance of the workers during the workshops of an experiment round. The evaluation of previous experiments on crowd-sourced software inspection can be found in the papers [23] and [29].
A formal definition of the evaluated sets and the performance metrics for workshops are described next.

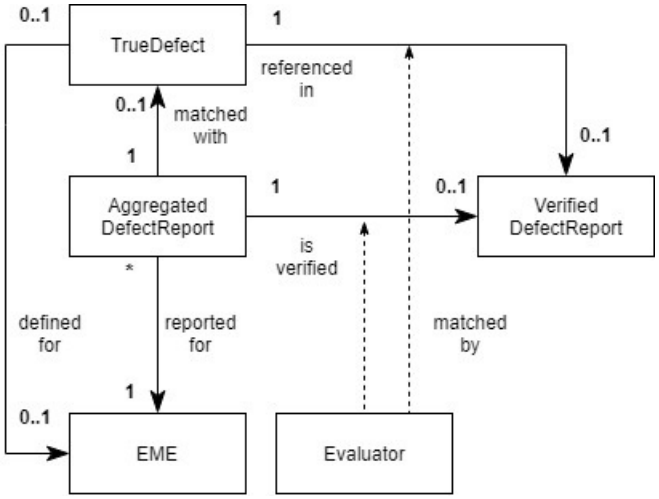


Figure 4.4: Evaluation data model.

Statistics for experiments

At the end of an CSI-Experiment (i.e. during VeriCoM) run, a number of *DefectReports* have been filed by the experiment participants. Not all of those reports state a defect in the model though, as the DefectReports of type *"NO_DEFECT"* describe a correctly modelled entity in the conceptual model. Also, after aggregation, a discord between workers on the type of a defect, which leads to no clear majority, aggregates to a DefectReport of type *"UNDECIDEABLE"*. Both these types of defects, need to be filtered out from the evaluation and lead to a set of clearly stated defects by the workers. This total set of AggregatedDefectReports (ADR) for a workshop is defined as $totalDefects$:

$$totalDefects = \{a \in ADR | a.type \neq NO\_DEFECT \wedge a.type \neq UNDECIDEABLE\} \tag{4.20}$$

Note that the above defined set, doesn't give information about the correctness (i.e. a matching TrueDefect exists) of the reports in the set. Therefore, the set of *TruePositives* (TP), i.e. AggregatedDefectReports within the set of *totalDefects*, which have a matching TrueDefect (TD) of the same type, is formalized as:

$$TP = \{a \in totalDefects | \exists t \in TD \text{ s.t. } a.emeId = t.emeId \wedge a.type = t.type\}. \tag{4.21}$$

Equally the set of *FalsePositives* FP, is defined as the complementary set

$$FP = totalDefects \setminus TP. \tag{4.22}$$

Analysing these sets provides insight into the capability of the workers to correctly identify defects in a conceptual model and are used as input for the overall workshop

performance metrics. For every experiment each of the held workshops is evaluated separately and then in combination, to see if there exists evidence for the correlation between the number of judgements and the number of undecided defect, true positives and false positives.

Workshop performance metrics

Two performance metrics, precision and recall, have been defined for a workshop. Workshop precision p_{ws}, i.e. the proportion of *TruePositives* within a workshop (TP_{ws}) to the reported defects in ($totalDefects_{ws}$) is formalized as:

$$p_{ws} = \frac{|TP_{ws}|}{|totalDefects_{ws}|}. \tag{4.23}$$

Computation of the recall value of TrueDefects within a workshop, requires the number of distinctly mentioned TrueDefects ($matchedTD_{ws}$) within the set of *TruePositives*. This can be formalized as a subset of TP_{ws} which contains each corresponding TrueDefect exactly once, i.e., $matchedTD \subseteq TP$.

With the value *matchedTD* computed, the workshop recall r_{ws}, can be computed as the ratio between *matchedTD* and the overall number of TrueDefects *TD*, which where present in the scenarios of the workshop:

$$r_{ws} = \frac{matchedTD_{ws}}{|\text{TD}|} \tag{4.24}$$

4.2.4 Student Feedback Components and Algorithms

Automated feedback generation for students was elaborated in requirement R41 and R42 (defined in Table 3.6) to improve the learning outcome during the workshops.
Figure 4.5 shows the ER-model of the designed feedback-components. A Worker participates in a number of workshops and can receive feedback for those workshops. During a workshop a worker reports DefectReports (DR) and ValidatedDefectReports (ValDR) for the two task T1-ModelAnlysis and T2-DefectValidation respectively. These DR and ValDR are evaluated and incorporated in the received feedback. The FeedbackService manages the creation of Feedback, aligns the workshops and their participating workers and calls other services to compute statistical values like precision and recall (see Section 5.3 for more details).

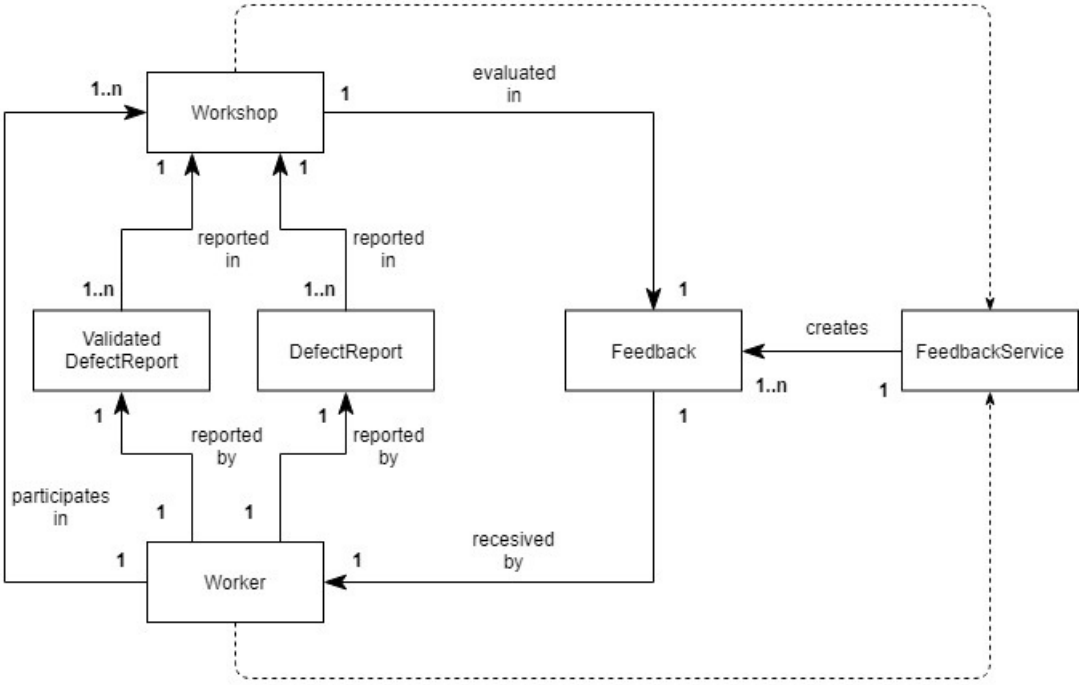


Figure 4.5: Feedback ER-model.

With all the needed statistical input computed and the feedback generated, the service to send feedback mails can be called, to deliver each participant valuable information about its contributions during the workshops.
The following sections will explain the formalization of the values, present in the feedback mail, in detail.

Correct Defect determination

As stated in Requirement R31 and R42 (see Table 3.5 and Table 3.6), there is a need for automatic determination of correctness of a reported defect. A *DefectReport* for an eme DR_{eme} is considered correct if there exists a TrueDefect defined for the same eme TD_{eme} with the same defect type, i.e., $DR_{eme}.type = TD_{eme}.type$. This set of correct defects matched with TrueDefects is describes as $M1$. The notion of correctness has been further expanded to additionally include defect reports with a different type as their corresponding TrueDefect ($M2$). Furthermore the neighbourhood relations of TrueDefect were also taken into account and a defect report was also considered correct if it mentions a TrueDefect within the TrueDefectNeighbourhood (see Section 4.1.4) of the actual TrueDefect in question ($M3$).

We define the set of correct defect reports CDR as follows:
Let DR be the set of all *DefectReports* reported by all workers of a workshop; let TD be

the set of all *TrueDefects* and let *TDN* be the *TrueDefectNeighbourhood* defined for a *TrueDefect t*;

The sets of matches between DefectReports and TrueDefects M1, M2 and M3, are defined as follows:

- M1: *DefectReports d* with the same *emeId* and *DefectType* type as a *TrueDefect t*.

$$\mathbf{M1} = \{d | d \in DR, \exists t \in TD \textbf{ s.t. } t.emeId = d.emeId \wedge t.type = d.type\} \quad (4.25)$$

- M2: *DefectReports* d with the same emeId as a TrueDefect t, without the reports of M1.

$$\mathbf{M2} = \{d | d \in DR \setminus M1, \exists t \in TD \textbf{ s.t. } t.emeId = d.emeId\} \quad (4.26)$$

- M3: DefectReports with the same emeId as an element n of TrueDefectNeighbourhood with relevance $rel = 1$, excluding the reports of M1 and M2.

$$\mathbf{M3} = \{d | d \in DR \setminus (M1 \cup M2), \exists n \in TDN \textbf{ s.t. } n.emeId = d.emeId \wedge n.rel = 1\} \quad (4.27)$$

A *DefectReport* $d \in DR$ is considered to be in the set of **CDR**, i.e.correct, when it is contained in one of the defined matching sets M1, M2, or M3:

$$\mathbf{CDR} = \{d \in M1 \cup M2 \cup M3\}. \quad (4.28)$$

Participant/Student feedback

To enhance the learning experience during an experiment run, workshop participants receive feedback with relevant performance metrics (requirement R41 of Table 3.6). These metrics consist of quality measures and group agreement values for the two distinct tasks *T1-ModelAnalysis* and *T2-DefectValidation*. The array of defined performance indicators, for the two tasks include:

1. T1 - ModelAnalysis:
   a) Reported defects / reported newly discovered defects
   b) Defect detection precision
   c) TrueDefect recall
   d) Group agreement on the type of a defect report
2. T2 - DefectValidation:
   a) Number of validated defect reports

b) Workshop average of validated defect reports

c) Group agreement on validation

d) Workshop average of group agreement on validation

These metrics are formally defined and further detailed in the following sections.

Feedback for T1-ModelAnalysis

For the task of model analysis an array of performance indicators have been defined. Quantitative measures like the number of defects reported, indicate whether the worker participated actively during the workshop or not. Outliers which contributed far below average, should not be considered in further experiment evaluation or weighted less than the rest of the workers.
New reported defects are defined as defects of an eme, for which no TrueDefect was defined and a textual defect description was given by the participants. This can either mean that the defect report is incorrect, or that the worker found a defect not anticipated by the task designers. These new defects need to be evaluated manually and the conceptual model needs to be checked for correctness.

Defect detection precision: Let DR_w be the set of defect reports filed by a worker w and let CDR_w be the set of correct defect reports defined in Section 4.2.4, with $CDR_w \subseteq DR_w$, then the defect detection precision for the worker p_w can be computed as the proportion:

$$p_w = \frac{|CDR_w|}{|DR_w|} \tag{4.29}$$

This precision value gives an indication of the participants understanding of model inspection and can be directly used to filter out under-performing workers when analysing and evaluating a workshop.

TrueDefect recall: The set of identified *TrueDefects* (i.e. a correct defect report was filed for the same EME) is defined as iTD_w and is a subset of TD ($iTD_w \subseteq TD$). The TrueDefect recall can then be computed as the proportion of the identified true defect of a worker, to all the true defects:

$$r_w = \frac{|iTD_w|}{|TD|}. \tag{4.30}$$

Additionally the set TD_w denotes the *TrueDefects* that where presented to the worker during model analysis. In other words, the worker was asked to inspect an eme for which a corresponding gold standard defect was defined.

$$r2_w = \frac{|iTD_w|}{|TD_w|}. \tag{4.31}$$

This additional definition refines computation of the recall and eliminates TrueDefects for emes which where not shown to the worker. Measuring these two different recall values gives exposure to the effectiveness and completeness of a software inspection carried out by a crowd of workers. Note that during the experiments of autumn 2017 (detailed in [23]) and spring 2018 only the recall value r_w was computed and considered for evaluation.

DefectType group agreement: With each given judgement, a defect report of a specific type is reported. Identifying the agreement within the group of participants on the type of the defect, gives insight into the group consensus about the given eme. Also, if agreement for a newly discovered unanticipated defect was high, it is a good indicator that the model has been designed with an error.

Let mDR be the defect report of the type which was reported by the majority of workers and let m_w be the group agreement value for a worker (majority value) for a single defect report. The set of defect reports for which a worker was in the majority regarding the defect type M_w, is defined as:

$$M_w = \{m_w | m_w \in DR, m_w.eme = mDR.eme \wedge m_w.type = mDR.type\}. \tag{4.32}$$

The respective percentage agreement value is computed as:

$$a_w = \frac{|M_w|}{|DR_w|}. \tag{4.33}$$

Feedback for T2-DefectValidation

The task *T2-DefectValidation* is used to determine, whether a given textual description of a defect from a previous experiment round, corresponds to a TrueDefect. To measure performance within this task, the number of validated defect reports is counted to measure worker activity during a workshop. For this task the agreement between workers plays a major role, as it impacts the final type of the defect report in question.

DefectValidation-Agreement algorithm: Algorithm 4.3 displays how it is determined, if a judgement of a worker was in the majority with the judgements of the group on the same task. As input values for the algorithm the ValidatedDefectReport a of the worker, needs to be supplied together with all the other reports V for the same task. Also the arrays for the two answer possibilities to the task questions and corresponding mappings to count the answers are needed as input. Output is a boolean value, indicating whether the worker gave the same answers as the majority or not (1).
First for each report given, the answers for each answer type are counted (2-13). This

part of the algorithm should be implemented separately and the resulting HashMaps should be saved and used further once they have been computed. Then the majority for the two questions is determined (14-15). If the worker answered with *IsTrueDefectAnswer.YES_TRUE_DEFECT* on the first question then he was also presented with a second question and he needs to be in the majority in both questions (16-19). Else it is sufficient if he is in the group majority for the first question (20-22).
The agreement value is then computed analogous to the agreement of task T1 in section 4.2.4.

Algorithm 4.3: DefectValidation-agreement algorithm

Input: A ValidatedDefectReport a, All ValidatedDefectReports for the same task V, An array of IsTrueDefectAnswers isTDAnswers, An array of DefectMeaningAnswers meaningAnswers, A HashMap<IsTrueDefectAnswer, int> answersTD, A HashMap<DefectMeaningAnswer, int> answersDM

Output: A boolean value isInMajority

```
1  isInMajority ← false;
2  for each v ∈ V do
3      for each i ∈ isTDAnswers do
4          if v.answerIsTD == i.answer then
5              answersTD(i) + +;
6          end
7      end
8      for each m ∈ meaningAnswers do
9          if v.answerDefectMeaning == m.answer then
10             answersDM(m) + +;
11         end
12     end
13 end
14 majorityIsTrueDefect ← answersTD.majority();
15 majorityDefectMeaning ← answersDM.majority();
16 if majorityIsTrueDefect == IsTrueDefectAnswer.YES_TRUE_DEFECT
   then
17     if a.anwerIsTD == majorityIsTrueDefect &&
       a.answerDefectMeaning == majorityDefectMeaning then
18         isInMajority ← true;
19     end
20 else if a.anwerIsTD == majorityIsTrueDefect then
21     isInMajority ← true;
22 return isInMajority;
```

In this section, algorithms for the CSI-Platform and VeriCoM are addressed. Specifically the algorithms for the conversion of FigureEight results into domain model entities, the

aggregation of defect types, the evaluation of experiment results and the creation of student feedback have been formalized and defined in pseudo-code. Also the performance metrics for the student feedback have been formally defined. These algorithms are part of the answer to research question RQ2 and build the basis for the implementation of the CSI-Platform prototype described in Section 5.

CHAPTER 5

Implementation

Building upon the classes and components defined in the design Chapter 4, a detailed description of the technology stack and software architecture with its subordinate modules (see Section 5.1) is given in this chapter. Furthermore a depiction of the VeriCoM-approach algorithm implementation, which forms the basis for the VeriCoM evaluation step, is provided in Section 5.2. In Section 5.3 the feedback service module is described. Also an overview of user interface elements of the CSI-Platform can be found in Section 5.4.

5.1 Architecture and technology stack

The software architecture of the CSI-Platform was designed as a layered 3-tier architecture, consisting of a controller-layer, a service-layer and a repository-layer. Each layer is only allowed to call the interfaces of the layer below to separate concerns [4]. Figure 5.1 shows the architecture of the CSI-Platform with its different modules, the connection to FigureEight, as well as the main technologies used for the implementation. The technology stack was taken over from *JHipster*[1], a development platform to generate *Spring Boot*[2] + *Angular* [3] Web applications. The application was intended to be written in *Java* where *Spring* is the go-to framework for dependency injection and fast development of the server-side of web applications. As a frontend framework to develop the User Interface (UI), *Angular* was chosen as it is easy to learn and offers the ability to structure the UI into components. *JHipster* allows to generate a pre-configured and well tested web application skeleton with a combination of those two frameworks, keeping additional configuration time to a minimum. *JHipster* follows a *Model-driven Software Engineering* (MDSE) approach, were the domain model of the application can act as a blue-print to generate basic code structures, e.g. model entities, repositories, services, REST-controller,

[1] https://www.jhipster.tech/
[2] https://spring.io/projects/spring-boot
[3] https://angular.io/

unit tests, etc., and a basic UI. Therefore *JHipster* was a good choice to generate the infrastructure of the web application and to focus on the development and implementation of algorithms for VeriCoM.

5.1.1 Code Modules:

The source code of the implemented CSI-Platform was split up into different modules (i.e. java packages), which group a self-contained set of classes and services together. The modules mirror the VeriCoM process steps and additionally separate some helper classes into their own module. Each module is composed of interface classes, which expose the offered services, and their respective implementation classes. This way the different architectural layers stay detached. The following modules have been created:

- **csvreaders:** Services to read in .csv-files, e.g. the results from FigureEight.
- **domain:** Domain model entities.
- **converters:** Converter-classes to convert non-domain objects, e.g. CSVRecords (i.e. one column of a .csv-file from FigureEight that represents multiple different domain objects), into their respective domain objects.
- **repository:** The repository layer of the architecture, offering several options to read data from the database.
- **dbpopulators:** Populator classes to populate the database with the converted domain objects.
- **api.figureeight:** FigureEight API client responsible for sending requests to the REST-API offered by FigureEight.
- **interpreter:** Interpreter classes to interpret results from VeriCoM and determining the defect type of an defect report based upon the answers given by the participant.
- **aggregation:** Services to aggregate DefectReports into AggregatedDefectReports (FinalDefects).
- **evaluation:** Services to match AggregatedDefectReports to their corresponding TrueDefect and to compute the performance metrics for each workshop.
- **feedback:** Services to provide feedback to students, i.e. computing the performance metrics for each worker and creating the feedback e-mail.

These modules assure a separation of concerns and structure the program in a way that is similar to VeriCoM. Note that the CSI-Platform is a prototype and changes in the requirements may result in the need to re-factor the modules. Details of the implemented classes which concern VeriCoM are presented in Section 5.2.

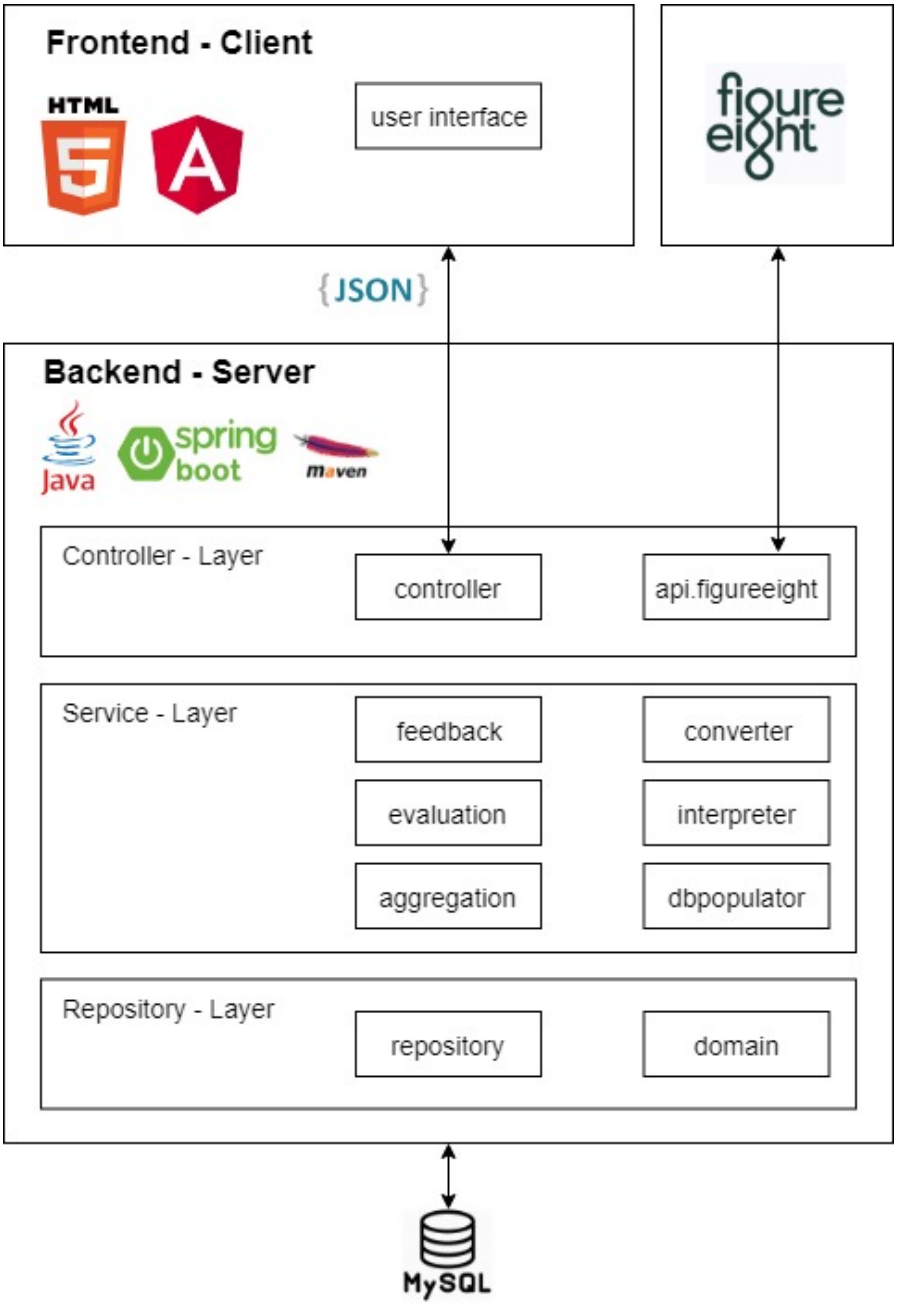


Figure 5.1: CSI-Platform implementation architecture.

5.1.2 FigureEight API-connection:

The connection to the FigureEight API is implemented as a Http-Client that performs REST-Request to the exposed FigureEight endpoints. The received Http-Response contains the requested resources in the JSON format [4]. The FigureEight endpoints allow to remotely create and manage crowdsourcing jobs, request monitoring information about the jobs, request worker judgements and the download of job-Results in the form of a .csv-file. The latter is used as the input for the VeriCoM algorithm implementation. It is also possible to configure a remote-hook, which sends live-updates of the jobs to the requesting client. This allows for a live-monitoring of worker performance during a workshop. This feature was not implemented in the CSI-Platform however, but should be considered for future work.

5.2 VeriCoM - Algorithm implementation

The generic VeriCoM approach, described earlier in Section 3.2, is the basis for the processes implemented in the CSI-Platform. The support for VeriCoM trough the platform developed in this paper, focused on the *Aggregation & Evaluation* part of the approach, which was identified as the most time consuming portion. The following subsections detail the implementation of this parts in the CSI-Platform.

5.2.1 Conversion & Interpretation - Modules

Preceding the actual VeriCoM steps aggregation and evaluation, specified in Figure 3.3, the algorithm implementation requires an additional conversion and interpretation step. Classes and interfaces created for this step are based upon the generic definition of Section 4.2.1 and are depicted in Figure 5.2. The Conversion & Interpretation modules address requirement R26 (see Table 3.3), which states the need to import .csv-files and the creation of domain objects.

Output data of the FigureEight-API is a .csv-file for each Job that has been worked on during an experiment. Each type of Job thereby produces it's own output file, which has to be distinctly converted and interpreted. The *"Results"*-classes are the first classes of the CSI-domain, containing the answers to the jobs *T1-ModelAnalysis* and *T2-DefectValidation* in their corresponding *"Judgement"*-classes, respectively. They are a direct representation of the answer given in FigureEight and are not interpreted, i.e. they do not have a defect type assigned to them yet. Each of the *"Results"*-classes is linked to the *Workshop* and *TaskInstance* in which they have been created. Converters and interpreters for *ModelAnalysisResults* and *DefectValidationResults* have been developed respectively. To interpret the given answers of task T1-ModelAnalysis into its correlating defect types, the *ModelAnalysisResultInterpreter* makes use of the *DefectTypeInterpreter* class. This *DefectTypeInterpreter* takes a *ModelAnalysisJudgement*, i.e. the answers to the guiding questions, and by means of Algorithm 4.1 determines a defect type and assigns

[4] https://www.json.org/

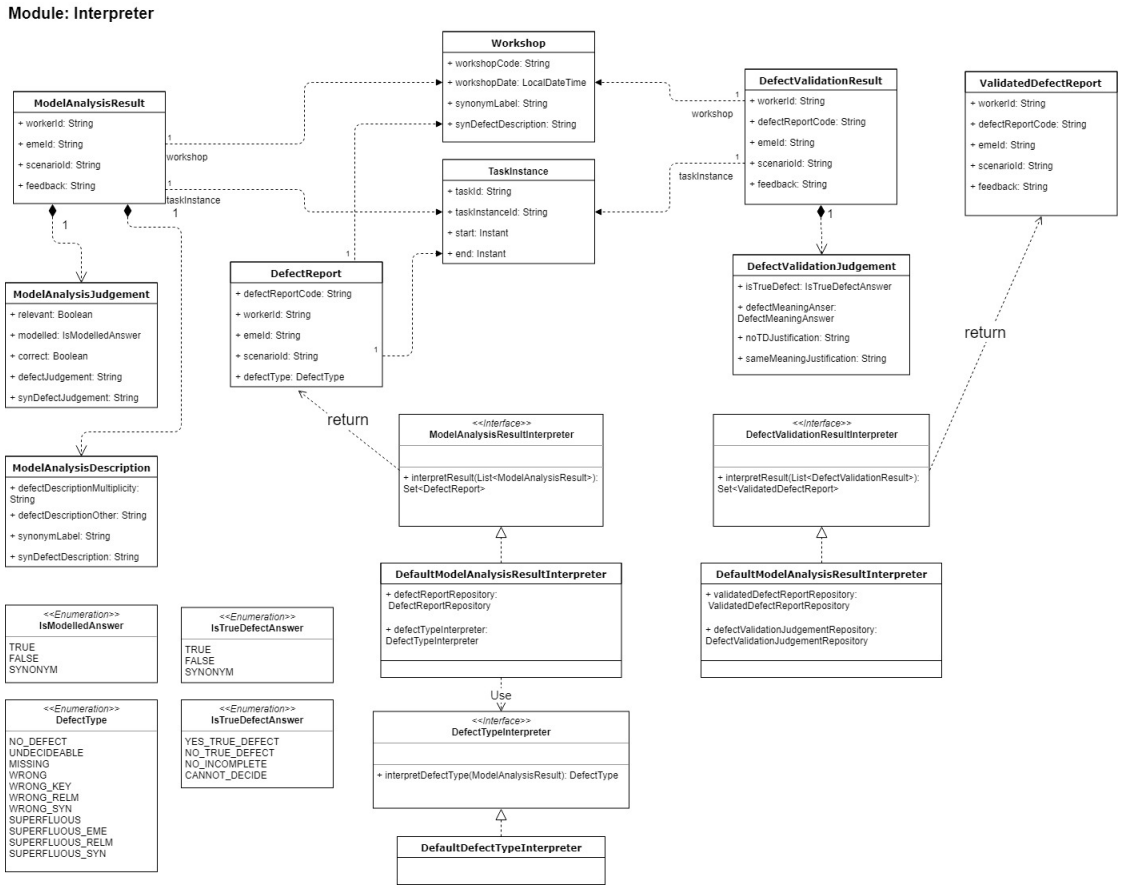


Figure 5.2: Interpreter-module class diagram.

it to the *DefectReport*. The *ValidatedDefectReport* of task T2, contains an interpretation of the questions that had been answered by the worker.

Outcome of the conversion and interpretation step, are *DefectReport*'s for *ModelAnalysisResult*'s and *ValidatedDefectReport*'s for *DefectValidationResult*'s. Furthermore, each column of the .csv-files from FigureEight, represents multiple entities of the CSI-Platform domain model. Therefore an interpretation of each .csv-column into their respective domain entities has to be performed. The interpretation of each of these entities is described in detail in the following sections. These entities function as input for the aggregation- and feedback-components in the following process-steps.

ModelAnalysisResult Interpretation

As depicted in figure 5.3 one of the human computation jobs, which is run in FigureEight during the CSI-Experiment, is Model Analysis. During the Model Analysis job, workers are presented a *Scenario*, a *ModelElement(ME)* as well as a conceptual model-diagram and they have to decide whether the ME is modelled correctly in the model. Guiding

their analysis is a series of questions further described in Section 3.2. Answers to the questions are represented as boolean values in case of a binary-choice questions or defined as Strings representing values of Enumeration-Types, introduced in Section 4.1.8, in case of more than two options to answer a question.

ModelAnalysisResults are downloaded from FigureEight as .csv-files (see Table 5.1), with columns containing the meta-data of the task together with the dataset of the question, e.g.: `task_id`, `taskinstance_id`, `worker_id`, `eme_id`, `scenario_id`, etc. and columns representing the answers to the guiding-questions e.g: `relevant`, `modelled`, `correct` .

Table 5.1: ModelAnalysisResult csv-record example.

task_id	**taskinstance_id**	**worker_id**	**eme_id**
t_12345	ti_12345	w_12345	DS62_Gr1_EME12 _EntA
scenario_id	**relevant**	**modelled**	**correct**
S02	true	true	true

The `ModelAnalysisResultConverter` takes this .csv-file as input and generates the following Java objects:

- `ModelAnalysisResult`: wrapper class, representing one task carried out by a worker
- `ModelAnalysisJudgement`: answers to the guiding-questions, i.e. the judgement about the correctness of the model element
- `ModelAnalysisDescription`: further descriptive text
- `TaskInstance`: the FigureEight task instance
- `Workshop`: the workshop in which this result was attained

In the consecutive interpretation step carried out by the `ModelAnalysisResultInterpreter`, `ModelAnalysisResults` and their respective `ModelAnalysisJudgements` are further refined. Taking a `ModelAnalysisJudgement` as input, the `DefectTypeInterpreter` defines the `DefectType`, determined by the answers to the guiding questions and the decision-tree in Figure 3.5. The interpretation of the judgements into the defect type is depicted in detail in Algorithm 4.1

Outcome of the interpretation of a `ModelAnalysisResult` is a `DefectReport` with a specific `DefectType`. These `DefectReports` represent the human computation task carried out by a worker and are further used in the aggregation- and evaluation-step and to provide feedback about the workers performance in the workshop.

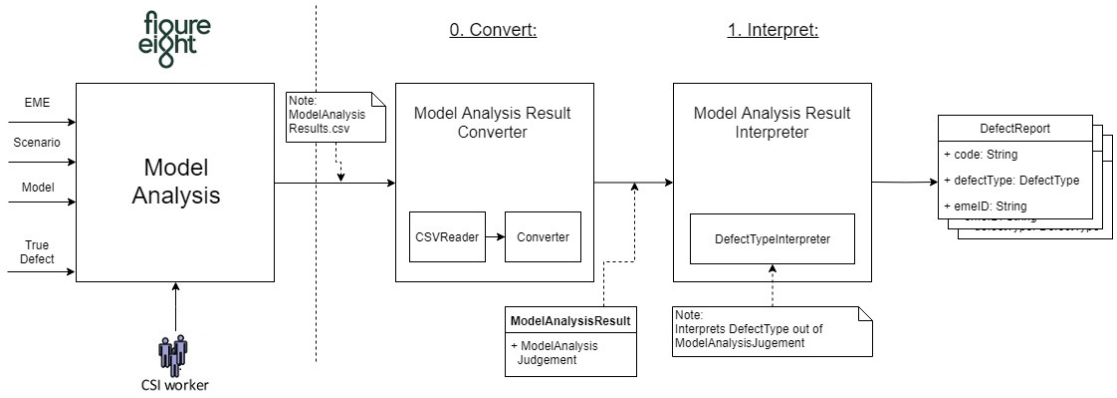


Figure 5.3: ModelAnalysisResult interpretation process.

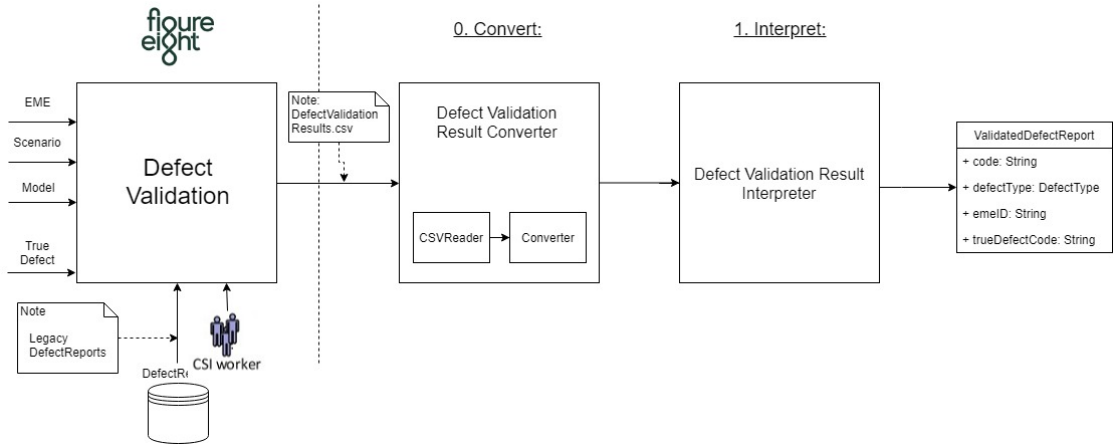


Figure 5.4: DefectValidationResult interpretation process.

DefectValidationResult Interpretation

Similar to the `ModelAnalysisResult`, the `DefectValidationResult` represents the outcome of the Defect Validation part of the CSI-Experiment. Figure 5.4 depicts the analogous conversion and interpretation process. In this second experiment task workers had to decide whether a given `DefectReport` from a previous experiment run has the same meaning as the corresponding `TrueDefect` of the same model element. The answers to these tasks are mapped by the `DefectValidationResultConverter` to the values of `IsTrueDefectAnswer` and `HasDefectSameMeaningAnswer`. Also the respective `TaskInstance` is generated. Output of this interpretation step is a `ValidatedDefectReport` instance containing the code of the initial `DefectReport`, the matched `TrueDefect` and the `DefectValidationJudgement` of the worker. `ValidatedDefectReport`'s are then further used to provide feedback about the agreement among workers judging the same defect report across one workshop.

5.2.2 Aggregation - Module

Requirement R32 (see Table 3.4), states the need for automated aggregation of *DefectReports*, to reduce the previous time intense process of manual aggregation. Therefore components for aggregation have been defined in Figure 5.5.

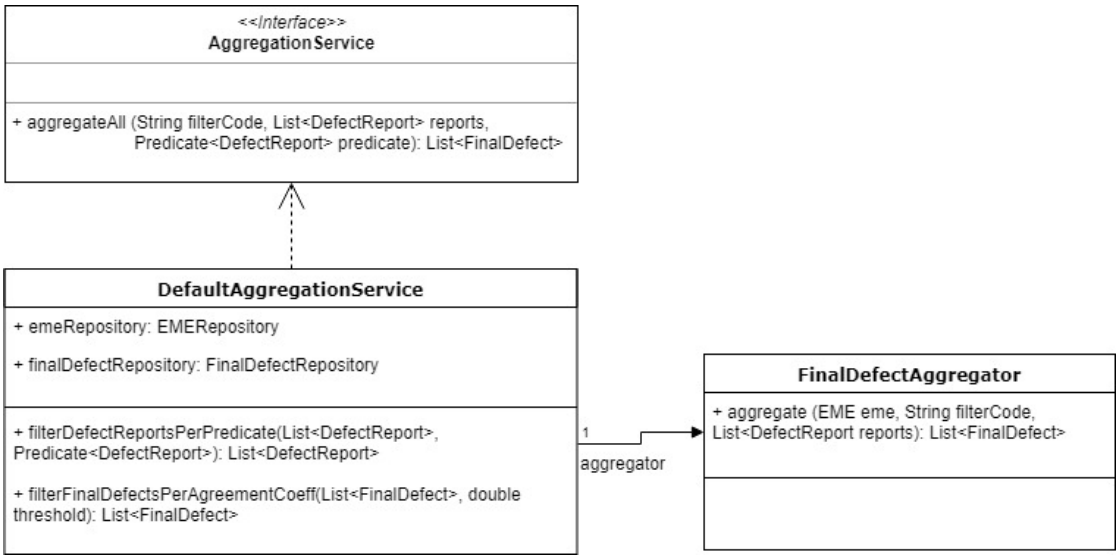


Figure 5.5: Aggregation-module class diagram.

The designed classes for the aggregation-module, consist of the interface *AggregationService*, its implementation the *DefaultAggregationService* and the *FinalDefectAggregator*. The *AggregationService*, encapsulates the core functionality, i.e. the method to aggregate all given *DefectReports* into *FinalDefects* with the previous application of filter predicates. The *DefaultAggregationService* streams over all emes defined in the *EMERepository* and calls the *FinalDefectAggregator*, to generate an aggregated defect report (*FinalDefect*) and to determine its aggregated defect type (see Algorithm 4.2).

To analyse the result of the task T1-Model Analysis, *DefectReports* reported for the same eme by n workers are grouped together and aggregated into a single `FinalDefect`, which represents the AggregatedDefectReports (ADR) formalized in Section 4.1.6, and contains the *DefectType* agreed-upon by the majority of workers who judged that eme. If there is a tie between multiple defect types, the FinalDefect gets assigned the `DefectType.UNDECIDEABLE`. The degree of agreement between workers on the type of a FinalDefect is denoted in the computed agreement coefficient. A detailed formal definition of the agreement coefficient and algorithm to determine the inter-rater agreement on the defect type is provided in Section 4.2.2.
As can be seen in Figure 5.6, a filter-predicate for the defect reports can be additionally passed to the aggregation service. This filter-predicate can be used to further refine the defect reports, for example to restrict aggregation to selected workshops. Predicates can

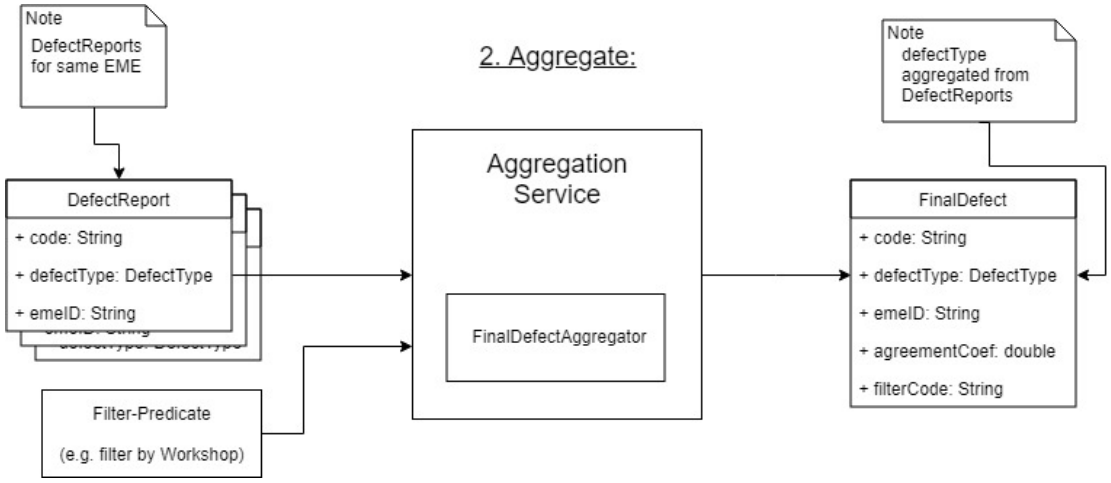


Figure 5.6: Aggregation of DefectReports to a FinalDefect.

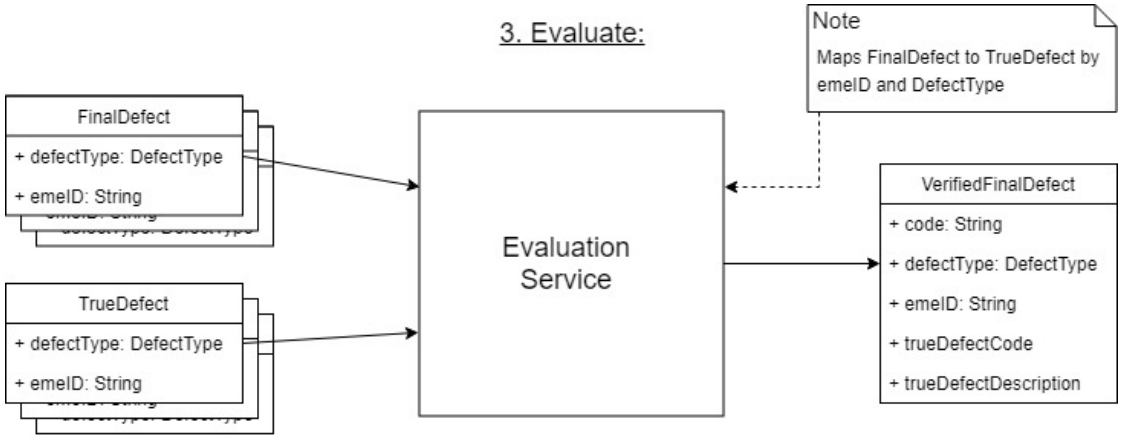


Figure 5.7: Evaluation of FinalDefects and TrueDefects to VerifiedDefectReports.

be defined as static-members of components, so they can be easily defined for common filter cases, e.g. filter defect reports per worker. These predicates can then be applied in other methods. The *"filterCode"*-field of a *FinalDefect* denotes the applied filter-predicate as a String.

5.2.3 Evaluation - Module

Evaluation of *FinalDefects*, aggregated in the Aggregation-step, happens by aligning them to *TrueDefects* (see Section 4.1.4), i.e. known defects in the model, defined for the same eme and with the same defect type (see figure 5.7). This alignment results in *VerifiedDefectReport*, which (in general) comprise a subset of *FinalDefects*.
The quality of the defect detection process (Model Analysis) is described by the two values, precision and recall. The first defines the proportion of matched VerifiedDefectReport

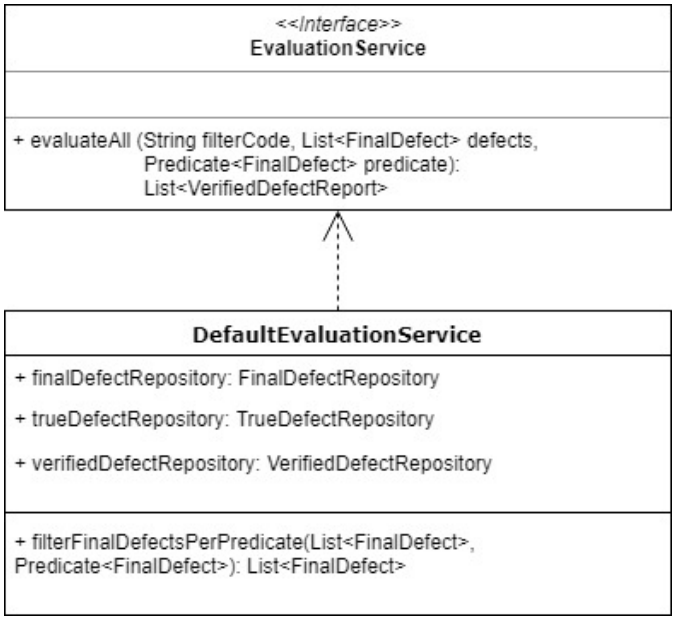


Figure 5.8: Evaluation-module class diagram.

to unmatched FinalDefects (also see Formula 4.23), while the later denotes the ratio of VerifiedDefectReport matched to distinct TrueDefects, i.e. the defects which could be identified in the model, over the total number of TrueDefects in the model (also see Formula 4.24). Together both indicate whether human computation yields promising results in model verification.

Figure 5.8 depicts the implemented classes for the evaluation module. The *EvaluationService* interface abstracts the implementation of the methods done in the *DefaultEvaluationService*. Within the *"evaluateAll()"*-method, the service tries to match the given FinalDefects with their corresponding TrueDefect (if present) and creates a set of VerifiedDefectReports. Filter predicates can be handed to the method to further refined the evaluation. Computation of performance metrics is handled by the *WorkshopStatisticService* within the feedback-module described in Section 5.3.

5.3 Feedback Service - Module

The components for providing automated feedback to the students participating in the experiment are grouped in the package *"feedback"* together with the services needed to compute the statistical performance metrics (see Figure 5.9). This module addresses requirements R51 and R52 described in Table 3.6.

Entry point to generate feedback, is the interface *"FeedbackService"*, which provides basic functionality to return feedback for all workshops, a specific workshop or a single participant. The *FeedbackService* is responsible for calling the services that compute

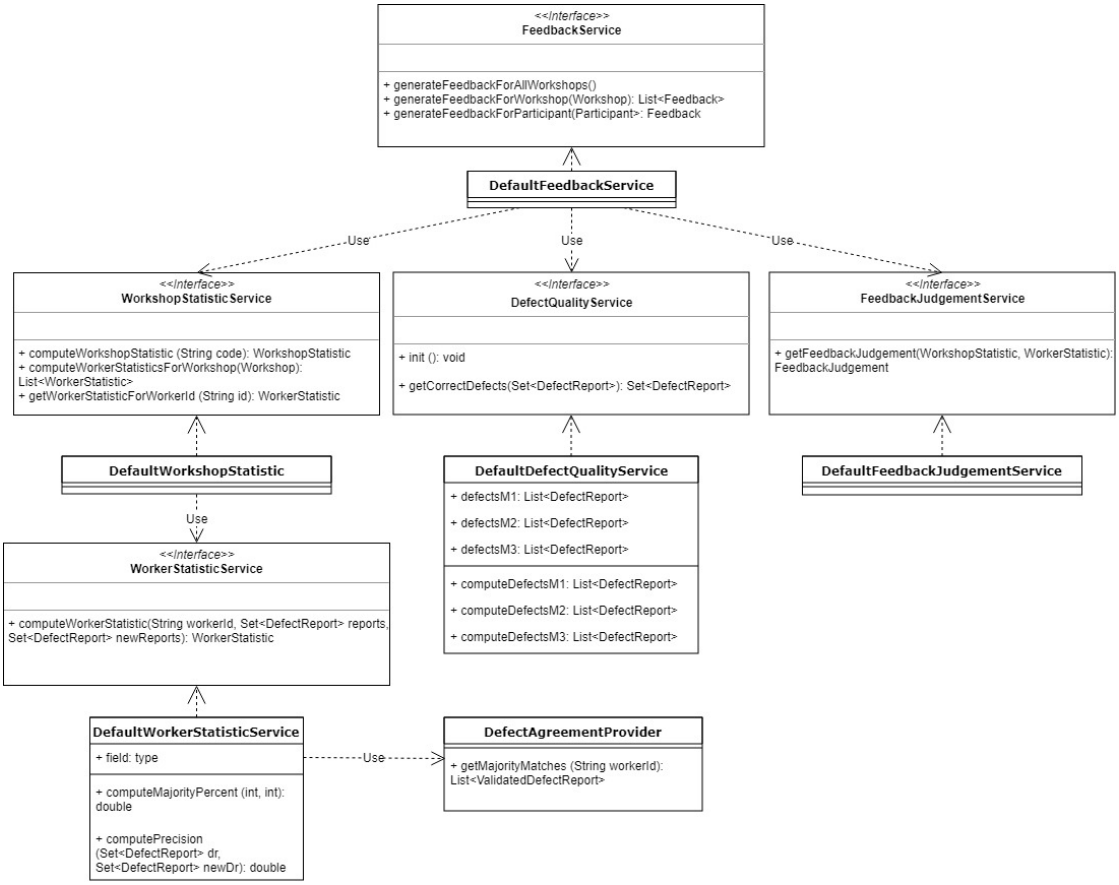


Figure 5.9: Feedback-module class diagram.

the statistical performance values for a given workshop and each participating worker. Given a *Workshop* instance, the *FeedbackService* creates a *Feedback* instance for each participant in that workshop. The service therefore delegates the computation of the needed values for an informative feedback to multiple other services, depicted in Figure 5.10. These supporting services have the following functions:

- The *WorkshopStatisticService* is responsible for computing the statistical metrics of a workshop. To be able to compute average values across a workshop, it needs to delegate the computation of worker specific performance values to the *WorkerStatisticService*.

- DefectReports and ValidatedDefectReports which share the same workshop-code are gathered from the database, are grouped by workshop-participant and sent to the *WorkerStatisticService*, which in turn uses provider-classes like the *DefectAgreementProvider*, to further delegate value computation.

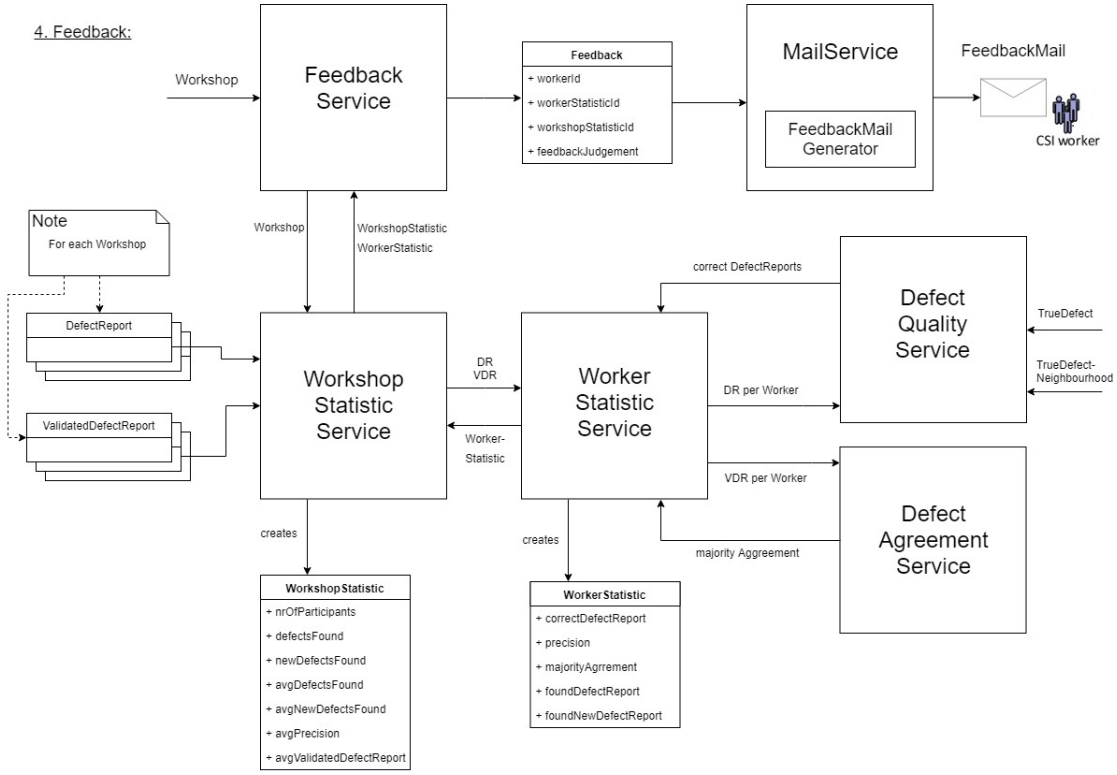


Figure 5.10: Feedback generation with Workshop- and Worker-statistic.

- The *DefectQualityService* is used to elaborate the correctness of a given set of defect reports. This algorithm is explained in detail in Section 4.2.4.
- The *FeedbackJudgementService* takes a computed *WorkshopStatistic* and *WorkerStatistic* and judges whether a worker performed above or below the average of the other participants in his workshop.

The computation of a *WorkerStatistic* is split into two parts, one for each part of the CSI-Experiment.

1. Model Analysis - defect detection precision: As a qualitative measure for Model Analysis, a precision value is calculate as the proportion of correctly identified defects of the worker over all his reported defects. To automatically determine which DefectReport was correct, the *DefectQualityService* creates three matching sets of DefectReports M1, M2 and M3, which are defined in Section 4.2.4. When a DefectReport of a worker can be found in one of those sets, i.e. it could be matched to a true defect or is in the neighbourhood of a true defect, then it is considered a correct defect report.

2. Defect Validation - defect validation agreement: ValidatedDefectReports from the second part of the experiment, are analysed whether the judgement of a participant correlates with the majority of judgements for the same defect report. As a result, a percentage value, representing the fraction of defect validation judgements, which align with the majority of the workshop participants, is returned.

All worker precision, recall and agreement values have to be computed in order to build averages over the whole workshop. Therefore, this step is computationally intensive and *WorkerStatistics* are only computed once and then persisted in the database.

With all the individual statistical values for a worker computed, also the overall averages of identified defects, newly discovered defects (i.e. erroneous elements of the model not anticipated by the experiment creators), worker precision/recall and validated defect reports for the workshop can be determined. Each participant is then given feedback based on his performance in relation to the average workshop result. These statistics are then further assembled in a *Feedback* instance, which is sent via the *MailService* to their respective CSI-workers.
The generated *WorkshopStatistic* is also displayed in the workshop-dashboard of the user interface (see Section 5.4).

5.4 User Interface

To address the requirements R11, R12 and R13 (see Table 3.2), a graphical user interface (UI) was developed to abstract from the database access via SQL-queries to interactions via the UI. The UI is suitable for users without SQL-database expertise and allows the whole experiment administration to quickly work with data models of the CSI-Experiments. The UI offers CRUD-functionality for all the entities of the data model defined in Chapter 4 and allows to export .csv-files for further analysis of the data (see Section 5.4.1). Furthermore, prototype overview dashboards have been developed to administer and analyse CSI-Experiments and workshops (see Section 5.4.2).

5.4.1 Data Model Visualization

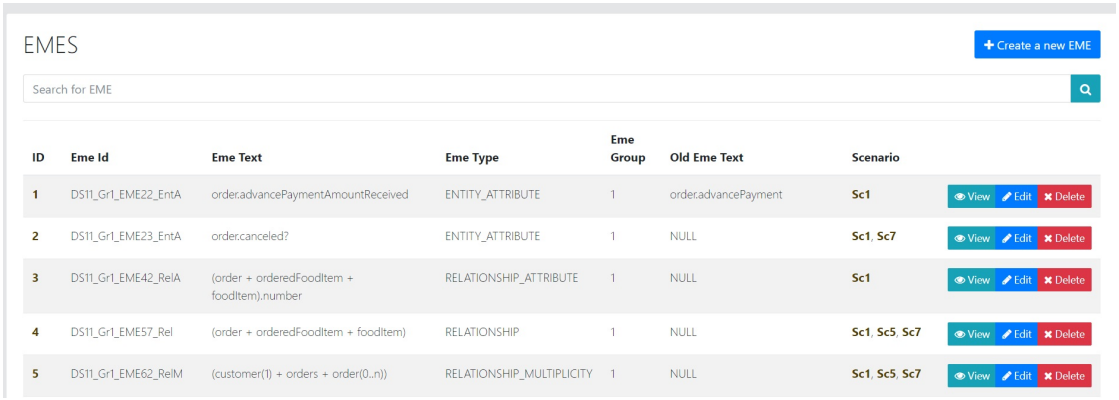


Figure 5.11: EME user interface.

To allow for customizing and editing of the CSI-Experiment data, the data model was visualized with a UI, which offers CRUD-functionality for each data model entity. Figure 5.11 depicts the UI for EMEs, showing their fields *"Eme Id"*, *"Eme Text"*, *"Eme Type"*, *"Eme Group"*, the *"Old Eme Text"* functioning as a reference to old data sets in case of changes, and a reference to the Scenarios in which the EME is mentioned. The reference to the Scenarios was especially important, because it allowed the experiment administrators to filter and create crowdsourcing jobs with all EMEs from a specific scenario and therefore reduced data preparation time.

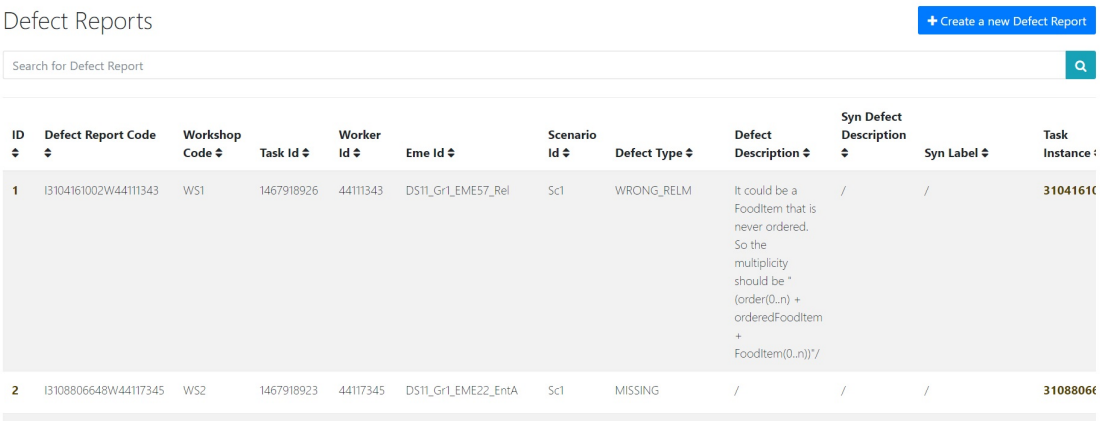


Figure 5.12: DefectReport user interface.

The UI for DefectReports is shown in Figure 5.12. It represents the answer of a worker to a single ModelAnalysis task within a crowdsourcing job. This view also allows to quickly change the DefectType if a revision of the DefectReport is necessary in case the worker created a DefectReport of type "WRONG".

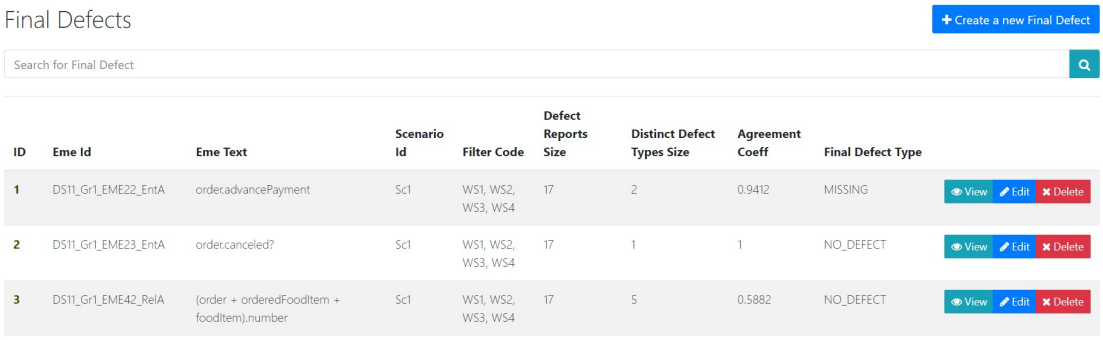


Figure 5.13: FinalDefect (AggregatedDefectReport) user interface

The FinalDefects (i.e., AggregatedDefectReports) table is displayed in Figure 5.13. Each row represents a FinalDefect with its corresponding fields, e.g. *"Eme Id"*,*"Eme Text"*, *"Scenario Id"* i.e., the EME and Scenario for which the FinalDefect was reported. *"Filter Code"* represents the codes of the workshops, which were taken into account to aggregate the FinalDefect. *"Defect Report Size"* is the sample size of DefectReports for the aggregation and *"Distinct Defect Types Size"* is the number of different DefectTypes within the samples. Finally *"Agreement Coeff"* is the rate of agreement between the workers on the reported defect and its type, i.e. *"Final Defect Type"*.

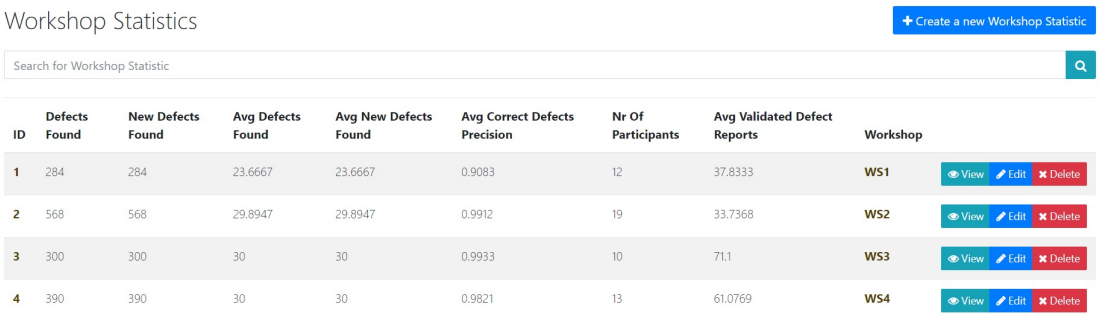


Figure 5.14: WorkshopStatistic user interface.

The UI for WorkshopStatistics data model is shown in Figure 5.14. The table displays the computed performance indicators, e.g. the number of reported defects, the average number of reported defects per worker, the average precision and the number of participants, for one or multiple selected workshops in a row.

5.4.2 Experiment Administration Dashboard

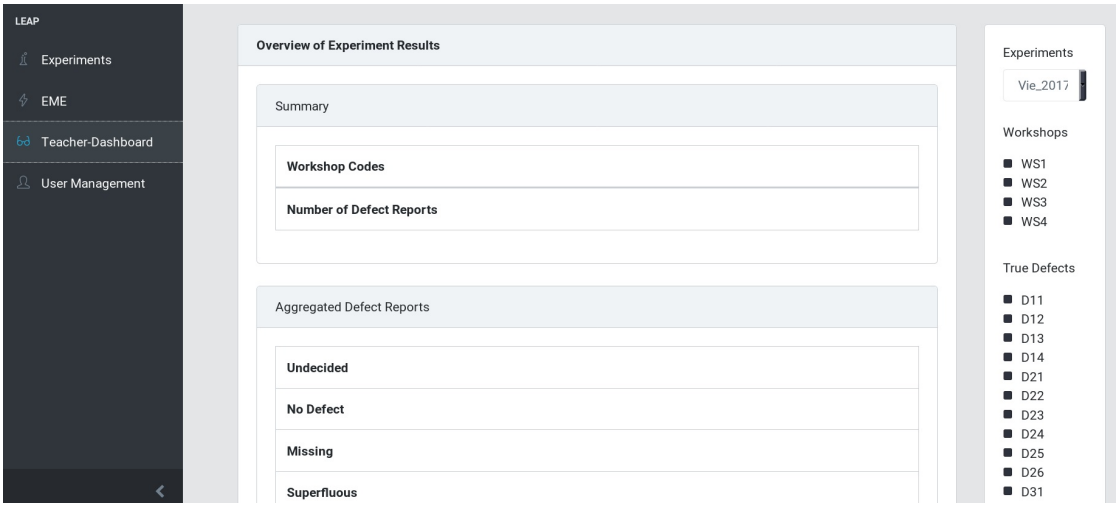


Figure 5.15: CSI-Experiment results overview.

For the CSI-Experiment administration team the evaluation of the experiment results was limited due to the lack of software tool support. The Administration dashboard (see Figure 5.15) offers an overview over the performed CSI-Experiments, the conducted workshops of that experiment and the accrued results. Note that the screen-shots already display an early prototype of the LEAP-UI which has been developed out of the CSI-Platform.

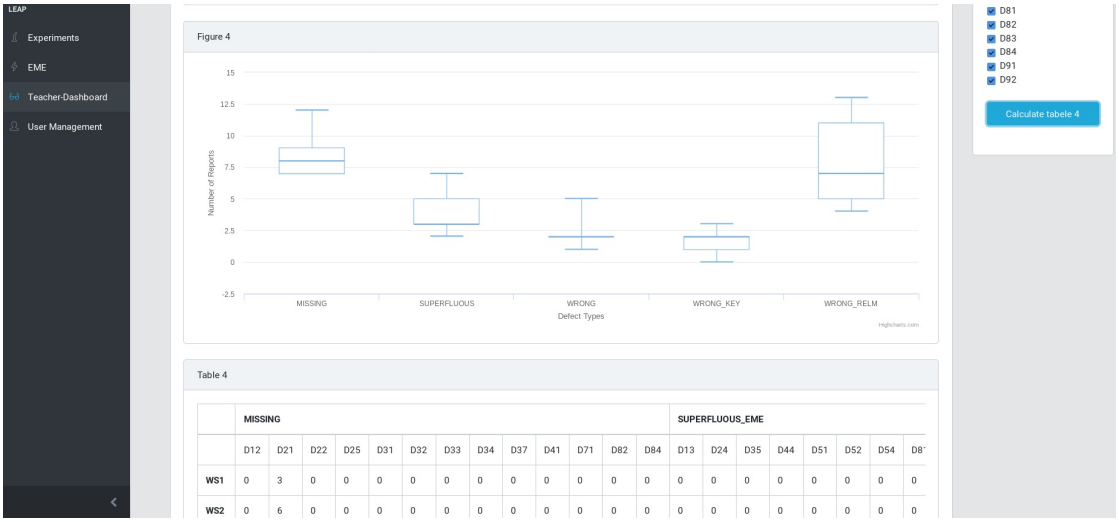


Figure 5.16: CSI-Experiment results.

Figure 5.16 shows a representation of the experiment results. The results can be filter

by workshop and by a selection of TrueDefects, which should be taken into account when computing the results. Also a hit/miss-matrix, representing the amounts a certain TrueDefect has been discovered within a workshop is offered in this overview.

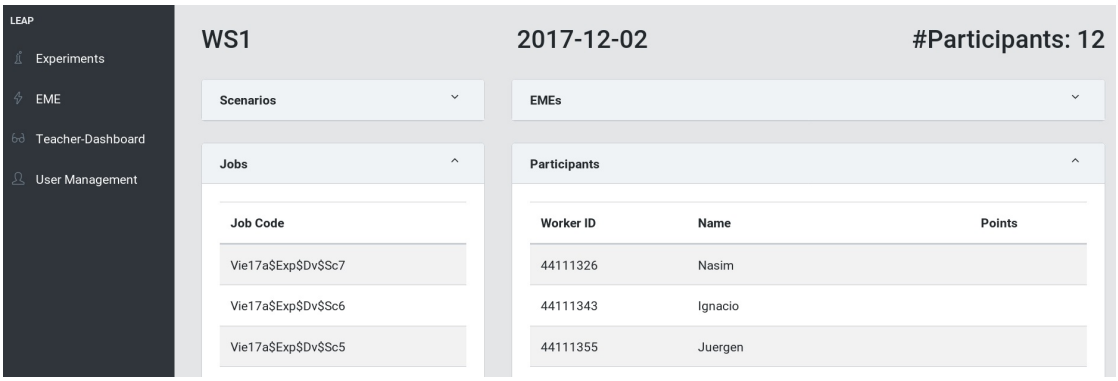


Figure 5.17: Overview over Workshop "WS1".

Figure 5.17 displays the meta-data of a workshop, including the Scenarios and EMEs the workshop was about, the crowdsourcing jobs that have been performed within this workshop, as well as the workshop participants. Overall the UI allows for a better insight in the structure of the CSI-Experiments and coherent access point for the administration team.

Based upon the defined data model and algorithms of Chapter 4, the corresponding implementation of the CSI-Platform prototype has been addressed in this chapter. First the technology stack and the architecture was described together with the tool used for rapid prototyping, JHipster. For each of the VeriCoM-steps, the developed classes and processes were explained in detail, addressing the created Services and their function. Finally important parts of the user interface have been shown to exemplify the look and feel of the CSI-Platform prototype. This prototype was used to support the CSI-Experiment of spring 2018 and the evaluation of this test run is presented in Chapter 6.

CHAPTER 6

Evaluation of the CSI-Platform prototype

With the CSI-Platform designed and implemented, the applicability of the prototype in the context of a live CSI-Experiment scenario was tested and evaluated in line with research question RQ.4 (see Section 1.2). Figure 6.1 depicts the time-line of the development of the CSI-Platform, from the starting point of the thesis up to the final evaluation workshop. The CSI-Experiment in which the first prototype was tested was the experiment of autumn 2017, which already showed promising results (see Section 3.4). Based on the insights that were gained through this test run, drawbacks from the previous approach have been identified and corresponding requirements have been derived to further refine the prototype and tailor it to the needs of future experiments (see Section 3.5 and Section 3.6). This refined prototype was then again tested and evaluated during the CSI-Experiment of spring 2018, where a thorough comparison of the improvements in the experiment process with respect to the previous approach (without software support) has been drawn in Section 6.2. The test setup of the CSI-Experiment of spring 2018 is described in Section 6.1 and details the application of the prototype during the Crowd-sourced Software Inspection.

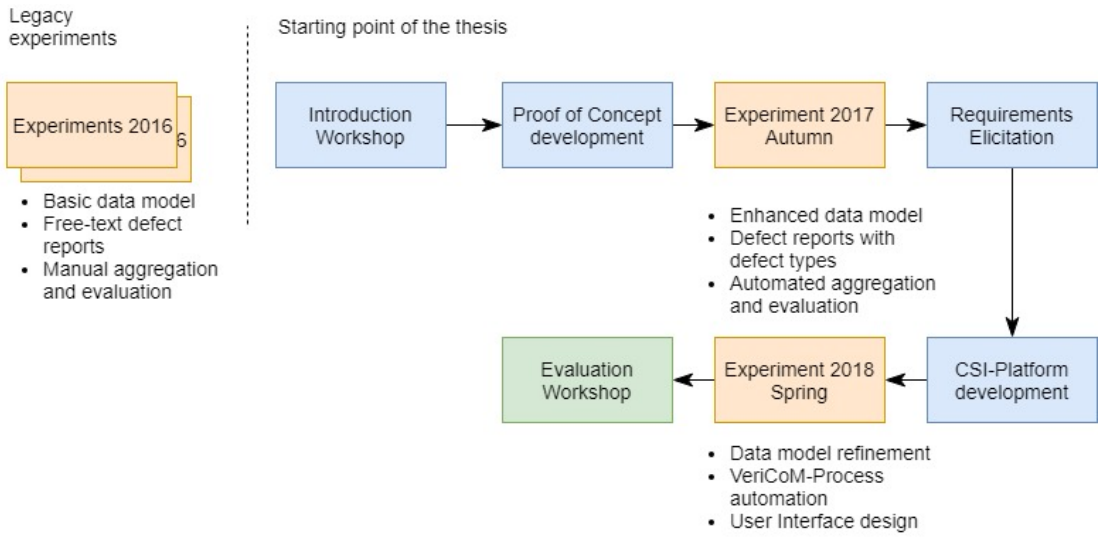


Figure 6.1: CSI-Platform Evaluation method and time-line.

After the last workshop of the experiment-run of spring 2018, a workshop for the experiment administrators was conducted, to clarify the improvements through the CSI-Platform for all the main stakeholders of the CSI-administrator group and to gather feedback for future improvements on the platform (see Section 6.3).
This feedback and suggestions for future improvements have been further used to draw a conclusion on the research question RQ.4 and to describe the envisioned Learning Analytics Platform (LEAP) in Section 7.3.

6.1 Experiment Spring 2018

The CSI-Experiments of Spring 2018 benefit from the support of the CSI-Platform. The experiment was conducted in an university setup with students participating as workers to perform the software inspection human computation tasks. The overall experiment setup consisted of the same tasks as the experiment of Autumn 2017, depicted in Figure 1.2a. The participants were split into two groups, the first performing the crowd-sourcing tasks, guided by the FigureEight crowd-sourcing engine, while the second group functioned as a control group, which performed classical software inspection with pen and paper. This control group, together with the control groups from previous experiments, function as a base line for the comparison of the classical approach and the VeriCoM approach.

The participant group which performed the crowd-sourced software inspection, had to perform the task *Model Analysis* (MA) and *Defect Validation* (DV). MA was improved through an automated suggestion concerning the defect type which the participant wants to report. This suggestion was made based on the answers to the guiding questions and eased automated aggregation of defect reports. DV, i.e. pairing a defect report with its

corresponding TrueDefect, was performed on legacy defect reports, which didn't have defect types assigned to them.

After each of the CSI-Experiment workshops, the results from the participants of the FigureEight (FE) human computation tasks were automatically downloaded, converted, interpreted, aggregated and evaluated. Each student received feedback and was informed about their performance through a generated e-mail.
The pen & paper group was evaluated manually and also received feedback via e-mail. Comparing the best-practice pen & paper approach and the VeriCoM approach, gave insight about the performance of a crowd vs. a single participant regarding: verification time and verification accuracy. The results of this experiment can be found in paper [23].

6.2 Comparison of unsupported experiment process vs. CSI-Platform supported process

Data preparation, result aggregation and evaluation of the CSI-Experiments before the experiment of autumn 2017, were manually performed tasks by the experiment administrators. These task were very time intensive, cumbersome and didn't scale for a larger group of participants. The developed software prototype provided functions to automate those tasks and additionally offered new functionalities to improve the overall process.
For each VeriCoM process step and the student feedback step, the identified drawbacks and requirements from Chapter 3 are used as guidance for the comparison. Comparison of the experiment process improvements through the CSI-Platform with respect to the previous unsupported approach is done by aligning the drawbacks (Dxx) of each step with the requirements (Rxx) that fixed them.
Note that the comparison doesn't aim to compare participant performance and does not give insight whether the VeriCoM-approach should be preferred over the classical pen and paper approach, but rather compares the unsupported manual CSI-Experiment evaluation process with its software supported counterpart.

- **VeriCoM 1.: Data Preparation** - The preparation of the crowdsourcing tasks for VeriCoM was a manually performed task. To gather the wanted data object, e.g. model elements, emes, scenarios, TrueDefects etc., SQL-knowledge was required to query the database for the entities. This meant writing difficult SQL-queries to filter the desired entities from the pool of the existing ones, which lead to a restriction of the user base to database-experts. As some of the experiment-staff were not database-experts, the workload for those who were increased. Some of the experiment-administrators therefore performed a workaround by filtering a generated .csv-file, containing all the entities, manually and copy&pasting together the selected entities.
  With the new visualized database in the form of a intuitive user interface (UI), this workaround is no longer necessary and the experiment-administrators can select

Table 6.1: VeriCoM 1.: Data Preparation improvements.

Experiment Autumn 2017	**Experiment Spring 2018: CSI-Platform**
D11, D12: SQL-expertise required to create datasets	R11, R12: Intuitive UI for easy dataset creation
D13: Manually filtered .csv-files	R13: Select and filter datasets through the UI
D14: Copy&Paste desired entities into .csv-file	R13: Automated .csv-file creation
D15: Data model dependent on FigureEight	R15: Independent generic data model with adaptable interfaces.

the desired data entities and create a .csv-file which can be directly uploaded to FigureEight to create crowdsourcing tasks (see Table 6.1).

- **VeriCoM 2.: Task Design and Execution (FigureEight interaction)** - In experiment runs previous to software support, the interaction with FigureEight was performed via the FigureEight web-interface. After each workshop the experiment-administrators had to manually download the results of each FE-Job (up to 15 jobs per workshop) and copy them into a .csv-file where it would the be later analysed. These files would then be interpreted, converted into domain object and persisted into the database by a staff member.
  With the CSI-Platform these steps were automated and generalized, so that it is now possible to use different crowdsourcing engines. Automation of FE-interaction, interpretation and conversion, lead to a significant reduction of time consumption, reduction of errors in the interpretation of results and overall less workload for the experiment staff members (see Table 6.2). Note that execution of crowdsourcing jobs and distribution to workers was still managed over FE.

Table 6.2: VeriCoM 2.: Task Design and Execution: Download, Interpretation and Conversion of FigureEight results.

Experiment Autumn 2017	**Experiment Spring 2018: CSI-Platform**
D25: Manual download of FigureEight results.	R24, R25: Automated download through UI.
D26: Manual interpretation of free-text defect reports.	R26: Defect type interpretation algorithm.
D26: Domain objects manually extracted from .csv-files	R26: FE results automatically converted into domain objects and persisted in database.

- **VeriCoM 3.: Aggregation** - Aggregation of defect reports was the part which benefited the most from software support (see Table 6.3). This part was performed manually, which made it time consuming, error-prone and not scalable for larger groups of participants. For each eme the corresponding free-text defect reports had to be gathered and the defect type reported by the majority of workers had to be aggregated. With the introduced defect types interpreted from the answers to the guiding questions, the aggregation was automated and is now scalable to a large set of input data.

Table 6.3: VeriCoM 3.: Aggregation improvements

Experiment Autumn 2017	**Experiment Spring 2018: CSI-Platform**
D31, D32, D33: Free text defect reports were manually aggregated.	R31, R32: Automated scalable aggregation of defect reports with defect types.
D34, D35: Inefficient and error-prone process	R33, R34: Multiple tested aggregation methods with filter options.

- **VeriCoM 4.: Evaluation and Experiment/Workshop Statistics improvements** - Evaluation of defect reports, i.e. aligning them with their corresponding TrueDefects and computing performance metric, was another VeriCoM process step relying heavily on manual work (see Table 6.4). Free-text defect reports were interpreted and matched with their presumed TrueDefect counterpart. With the introduction of defect types this step was automated to provide instant evaluation of defect reports and computation of performance metrics.
  Statistical performance values of a single workshop or an entire experiment run, had to be computed by manually collecting the desired data and analysing excel-spreadsheets. This task can now be performed with the UI, which offers a visualization of defined performance metrics like precision, recall and worker-agreement in various tables and graphs. Additionally the input data can be filtered to analyse only distinct workshops of an experiment run or combine different workshops to compare them.

Table 6.4: VeriCoM 4.: Evaluation and Experiment/Workshop Statistics improvements

Experiment Autumn 2017	**Experiment Spring 2018: CSI-Platform**
D41: Manual interpretation of free-text defect reports.	R41: Automated interpretation/evaluation.
D41: Matching TrueDefects to defect reports cumbersome and error-prone.	R41: Automated TrueDefect matching.
D42: Manual collection of experiment/workshop data.	R42: Intuitive UI, with filter options.
D42: Manual statistics computation via excel-spreadsheets.	R42: Algorithms for Automated computation of performance metrics; precision, recall averages, agreement etc. with filter options.
D43: Lack of visualization of the workshop performances.	R43: Workshop dashboard.

- **5. Feedback for students/participants** - Previous to the CSI-Platform, there was no feedback or performance indicators for students/workshop-participants, which limited their learning experience through the workshop.
  Now a detailed feedback-mail with different performance metrics for each of the workshop-tasks is generated and sent to the students and the platform offers an overview dashboard where they can compare themselves with the average performances of other workshop participants. This can enhance their learning outcome and can improve their model analysis skills.

Table 6.5: 5. Student/Participant feedback improvements.

Experiment Autumn 2017	**Experiment Spring 2018: CSI-Platform**
D51, D52: No performance feedback for participants.	R51, R52: Automated feedback-mail with performance metrics to enhance learning outcome.
D53: No possibility to review results.	R52, R53: UI with detailed performance overview and ability to compare results of other participants.

6.3 CSI-Platform Evaluation - Interviews

The evaluation of the CSI-Platform was done via an interactive workshop/interview session with the experiment-administrators. Within the course of this workshop, the functionality offered by the CSI-Platform was explained to the experiment-administrators in the from of presentation slides and a live demo of the platform's user interface. Each

step of the experiment process and its supporting features of the software prototype have been explained in detail, followed by a questionnaire about the current process-step to determine the perceived improvement.

Note that the goal of this workshop was primarily not to gather a quantitative score on the stakeholders perceived satisfaction with the CSI-Platform, but rather to foster discussion about the prototype and to clarify a common vision for the **LE**arning **A**nalytics **P**latform (**LEAP**) which is planned to be developed in the future (see Section 7.3).

6.3.1 Interview Method and Question Structure

The questionnaire was comprised of one section for each of the CSI-Experiment process/VeriCoM steps. The interviewees witnessed a detailed presentation about the supporting feature of the software prototype before each section and then had time to answer the questions within this section and give detailed feedback. For each process-step/stage the stakeholders were asked to give their assessment on:

- *"How import was software support for this stage?"* (Sx_Q_I),
- *"How satisfied are you with the software support?"* (Sx_Q_S),
- *"What are possible future improvements/extensions?"* (Sx_Q_F).

Each question was given a code based on their section Sx and the type of question, i.e. Q_I for the importance of software support, Q_S for the satisfaction with the support and Q_F for suggested future improvements. The first two questions had to be rated between 1 (*not important/not satisfied*) and 5 (*very important/very satisfied*), followed by a text filed to specify their desired future improvements/extensions to the platform.

The complete questionnaire is depicted in the following sub-sections in the way, that first the questionnaire section with its corresponding descriptions is shown, together with the received answers, followed by the suggested future improvements.

6.3.2 S1 - Modelling CSI-Experiment Domain

The first section about modelling the CSI-Experiment domain marked an exception in the section structure, with an additional question before the ones mentioned above. The stakeholders first had to answer: *"How did the definition of the CSI-experiment domain model help to clarify the overall experiment structure? (If applicable please specify.)"* (S1_Q_D), with a score between 1 representing *"All the parts were already known."* and 5 *"There were a lot of parts which have now been clarified."*. Also a subsequent textual specification to their answers was requested. This question had the purpose to draw attention to the domain model design and the naming conventions defined for the CSI-experiment process, as it lacked a clear definition of names and types, which lead to misunderstandings. Some of the stakeholders didn't have insight in the domain model therefore this question was optional.

Table 6.6: Answers to S1 - Modelling CSI-Experiment Domain.

Question	**Clarification/Importance/Satisfaction**
S1_Q_D	For 3 interviewees, the domain model definition clarified parts of the overall experiment structure, while 1 claimed that most parts are now clearer.
S1_Q_I	This step was considered to be important and very important by two stakeholders each.
S1_Q_S	The satisfaction level was equal to the importance with two stakeholders stating satisfied and very satisfied each.

The first core task before starting the implementation of the support platform and conducting the CSI-Experiments of autumn 2017, was the creation of an in-depth model of the experiment domain, i.e., creation of a domain model and relations, definition of DefectTypes and decision tree, definition of Enumeration types (see Section 4.1). This also includes the relation between all the stakeholders and systems involved (Tuwel, FigureEight/CrowdFlower, CSI-Platform, database, evaluation-paper etc.).

Questions:

Table 6.6 shows the results of the questions about the designed CSI-Experiment domain model. The given answers are discussed in detail below.

- S1_Q_D: How did the definition of the CSI-Experiment domain model help to clarify the overall experiment structure? (If applicable please specify.)

  Specification on S1_Q_D:

  - *"Most of the entities became clearer in the experimental team."*
  - *"Overall experiment process is now well-structured."*
  - *"Some of more in-depth concepts can still be added, but the major concepts were well captured."*
  - *"Clarification of experiment data aggregation step."*

  The answers to Question S1_Q_D together with their textual specification, show that the designed domain model and the structural naming conventions, clarified the overall concepts and entities used in the CIS-experiment context. It was also mentioned that communication within the experiment administration team was improved and misunderstandings were reduced. This answers can be directly linked to research question RQ1 and show that a clear definition of the domain model improves the CSI-experiment process overall.

- S1_Q_I: How important was support for this stage? Supporting the domain model design was seen as important even if some of the parts of the domain model were

already defined a in depth refinement of the existing entities and the relations between them was well-received.

- S1_Q_S: How satisfied are you with the support? The overall domain model presented to the stakeholders shed light on some newly desired features and encouraged a more structural approach to new CSI-experiment rounds, with the definition of a domain model for future experiment use cases.

- S1_Q_F: What are possible future improvements/extensions?
  Structuring the domain model with certain new experiment use cases in mind, was mentioned as a future improvement for this design phase. Also a distinct separation of the data model and their processing components was requested. It was also stated that the capturing of synonyms within the conceptual model is a desired feature. While synonymous model elements are already captured there have been no evaluation functions defined for them to separately evaluate them. The automated creation of crowdsourcing job was another request from the stakeholders.
  During the presentation the question arose, whether it would be possible to map the old experiment data, as well as the defect reports captured with the pen and paper approach, to the current data model. This is technically possible as the current design of defect reports can also be applied for the pen and paper approach. To achieve the mapping a defect report has to be created and the manually given description has to be interpreted (with the help of human computation) and the according defect type has to be assign. Then those defect reports can be evaluated in the same way as the ones created through crowd sourcing.

6.3.3 S2 - Data Preparation - VeriCoM Step 1

In this step the CSI-Platform helped to assign EMEs to Scenarios, to create TrueDefects, to visualize data and to allow for the export of .csv-files that could be uploaded to CrowdFlower/FigureEight.

Questions:

Table 6.7 shows the results of the questions for the VeriCoM Data Preparation step. The given answers are discussed in detail below.

- S2_Q_I: How important was support for this stage? The importance for software support for the data preparation step was considered very high, as previously this step could only be performed through direct database access via SQL-queries.

- S2_Q_S: How satisfied are you with the support? The stakeholders were somewhat satisfied with the offered support for this step. This could be due to the fact that the user interface was not final during the course of the CSI-Spring-Experiment and the preparation of data was done by the author on request of the stakeholders.

Table 6.7: Answers to S2 - Data Preparation.

Question	Importance/Satisfaction
S2_Q_I	The importance of the data preparation step was highlighted and was considered very important by 3 main stakeholders and important by another.
S2_Q_S	Satisfaction with the offered solution ranged from high to medium. This could be due to the fact that the UI was not final during the time of the experiment.

Table 6.8: Answers to S3 - Task Design and Execution.

Question	Importance/Satisfaction
S3_Q_I	This step was considered very important by 3 of the main stakeholders, while one did consider it somewhat important with the explanation that he current FigureEight platform does an acceptable job.
S3_Q_S	Three interviewees were satisfied and one was very satisfied with the offered support through the CSI-Platform.

- S2_Q_F: What are possible future improvements/extensions?
  As Question S2_Q_S already captured, the improvement of the user interface, to make data preparation available for the experimental team and not just experts. This was again mentioned as a desired feature for the next iteration of the software prototype. Versioning, automated consistency checks, automated creation of crowdsourcing jobs and a more flexible creation of new model elements and emes to incorporate different kind of inspection approaches, were also requests for the future work on this topic.

6.3.4 S3 - Task Design and Execution - VeriCoM Step 2

In this step the CSI-Platform supported the automatic download of finished jobs and participant data, and provided a definition for a generic domain model for the connection to FigureEight. (Job execution and Task design was out of scope for the first prototype.)

Questions:

Table 6.8 shows the results of the questions for the VeriCoM Data Preparation step. The given answers are discussed in detail below.

- S3_Q_I: How important was support for this stage? As the created prototype in this paper doesn't cover crowdsourcing task design and job execution, FigureEight was used for this process step. The experiment team however appreciated the automatic download and saving of finished job reports and participant data.

Table 6.9: Answers to S4 - Interpretation and Conversion.

Question	Importance/Satisfaction
S4_Q_I	This step was considered to be in need of software support, with 3 experiment administrators stating high importance and 1 stating importance.
S4_Q_S	The offered support was received as very satisfying and satisfying by 2 stakeholders each.

- S3_Q_F: What are possible future improvements/extensions? The replacement of FigureEight as the crowdsourcing engine was considered a future improvement and is further described in Section 7.3.

6.3.5 S4 - Interpretation and Conversion - Intermediate step

In this intermediate step the CSI-Platform automatically interpreted a DefectType based upon the answers given to the guiding questions by a workshop participant and the raw-data (.csv-file) from FigureEight was converted into DefectReports.

Questions:

Table 6.9 shows the results of the questions for the intermediate Interpretation and Conversion step. The given answers are discussed in detail below.

- S4_Q_I: How important was support for this stage? The intermediate step for interpretation and conversion of FigureEight domain entities into domain model elements was considered very important, as there now is a clear separation of those two domains.

- S4_Q_S: How satisfied are you with the support? The generic implementation of the interpretation and conversion step was satisfiying although the implications for future iterations of the prototype could not directly be shown in practice. However it was acknowledged, that this conversion-layer was very important for to allow for aggregation and evaluation and is a core component for future iterations of the software.

- S4_Q_F: What are possible future improvements/extensions?
  In the future LEAP platform, the participant data has to be anonymized and be conform to the EU General Data Protection Regulation (DSGVO [1]). This means that it will be no longer possible to use FigureEight as it would expose personal student data to a third party, therefore an adaptable conversion layer built upon the current infrastructure was considered as a future improvement.

[1] https://www.iitr.us/eudatap.html

Table 6.10: Answers to S5 - Aggregation.

Question	Importance/Satisfaction
S5_Q_I	Supporting the Aggregation step of VeriCoM was unanimously seen as very important.
S5_Q_S	Three of the main stakeholders were very satisfied with the support offered by the CSI-Platform and one stakeholder was satisfied.

6.3.6 S5 - Aggregation - VeriCoM Step 4-1

In this step the CSI-Platform provided automatic aggregation of DefectReports into AggregatedDefectReports (FinalDefect) with the aggregation of DefectTypes based on majority rating.

Questions:

Table 6.10 shows the results of the questions for the Aggregation step of VeriCoM. The given answers are discussed in detail below.

- S5_Q_I: How important was support for this stage? Aggregation of experiment data was a manual and very time consuming error prone task and was therefore considered in desperate need of automation through software support.
- S5_Q_S: How satisfied are you with the support? The experiment team was very satisfied with the implemented aggregation algorithm and the replacement of manual aggregation.
- S5_Q_F: What are possible future improvements/extensions?
  Live-feedback, i.e. UI elements showing the aggregation during the workshop runs was a desired extension of the current functionality of the aggregation module, with different views on the performance of the participants, e.g. a 0/1-matrix depicting the currently reported defects which correspond to true defects. Also new sorting functionality, e.g. sorting by ascending or descending ACoeff-values and the flexible definition of aggregation strategies, was mentioned together with filtering out under-performing participants.

6.3.7 S6 - Experiment Evaluation - VeriCoM Step 4-2

In this step the CSI-Platform provided automatic evaluation of DefectReports, (i.e. matching with TrueDefects, generation of VerifiedDefectReports), statistical evaluation of experiments, computation of performance metrics.

Questions:

Table 6.11 shows the results of the questions for the Evaluation step of VeriCoM. The given answers are discussed in detail below.

Table 6.11: Answers to S6 - Evaluation.

Question	Importance/Satisfaction
S6_Q_I	Supporting the Evaluation step of VeriCoM was unanimously seen as very important.
S6_Q_S	Three of the main stakeholders were very satisfied with the automated experiment evaluation methods, offered by the CSI-Platform and one stakeholder was satisfied.

Table 6.12: Answers to S7 - Student-Feedback.

Question	Importance/Satisfaction
S7_Q_I	Providing feedback to students was stated to be very important by two stakeholders and important by the other two
S7_Q_S	The developed solution satisfied three of the main stakeholders with one being very satisfied.

- S6_Q_I: How important was support for this stage? The evaluation of the experiment results (i.e. matching defect reports with true defects) and the computation of performance metrics, was another core task which was previously performed manually and was very error prone. Software support was therefore considered very important.

- S6_Q_S: How satisfied are you with the support? The experiment team was very satisfied with the implemented features and the offered software support.

- S6_Q_F: What are possible future improvements/extensions?
  For the evaluation new evaluation metrics have been considered as future improvements and also the before mentioned 0/1-matrix should be a feature in the LEAP platform specification. Also the false positives for a set of defect reports should be analysed.

6.3.8 S7 - Student-Feedback

The CSI-Platform supported the automatic generation of Student-Feedback-mails with performance metrics such as: precision, recall, workshop averages, workshop agreement on defect type.

Questions:

Table 6.12 shows the results of the questions on the feedback for students which was automatically created by the CSI-Platform. The given answers are discussed in detail below.

- S7_Q_I: How important was support for this stage? Feedback for students was a completely new functionality and was mentioned as important, while it doesn't directly affect the experiment outcome but is an important benefit for the participating students.
- S7_Q_S: How satisfied are you with the support? The newly implemented feature was satisfying as a first draft of the feedback mechanisms and can be seen as a baseline to build upon for the LEAP platform.
- S7_Q_F: What are possible future improvements/extensions?
  To increase the learning outcome of the participating students, various graphical representations of their performance were describe for future improvements. This includes live feedback and the scope of tasks the student worked on, as well as considering the TrueDefectNeighbourhoods in the defect analysis together with a depiction of all the correct and incorrect judgements of the worker. Also a flexible description of parameterized student feedback e-mails was desired.

The evaluation of the CSI-Platform showed that by defining common naming conventions and a concise data model, VeriCoM and the communication between the experiment administration team as a whole improves. The support for each of the VeriCoM steps, that was offered by the CSI-Platform, was broadly well received. Especially the automation of the most time consuming steps of VeriCoM, i.e. aggregation of defect reports, evaluation of the experiment workshops and provision of student feedback, were mentioned the most beneficial for the experiment team. This verifies the research question RQ4 and shows that the CSI-Platform can act as a prototype for further improvements on supporting VeriCoM.

CHAPTER 7

Conclusion and Future Work

Building upon the works and publications of Sabou, Winkler et al. [23, 29, 30, 31], a thorough investigation of Software Inspection (SI) processes, in particular the improvement of traditional best-practice approaches through Human-Computation and Crowdsourcing (HC&C) techniques has been conducted. Literature research has been performed to illustrate the emergence of Crowdsourced Software Inspection (CSI) out of SI by application of HC&C techniques.
For the generic problem of the **Veri**fication of **Co**nceptual **M**odels, the **VeriCoM**-approach was presented and applied on a Software Engineering use case. To motivate the importance of VeriCoM and to highlight the benefits of crowdsourcing applied in this domain, experiments have been performed to verify this approach.
These experiments followed a strict scientific experiment-process, which was mostly performed manually and was very time-consuming, error prone, not scalable and overall not applicable for a larger crowd of participants. Therefore the need for a software tool to support this experimental process arose. This thesis focused on the detailed formalization of the data model and algorithms to automate the VeriCoM-steps and to implement a prototype for such a software support tool, the CSI-Platform.

7.1 Answers to Research Questions and Discussion

The development of the CSI-Platform prototype was performed in consecutive software development process steps. In the following, the contributions and results of this work to the respective research questions are discussed, together possible limitations and lessons learned:

- ***RQ1:*** *Which parts of the Crowdsourced Software Inspection experiment process benefit the most from software tool support?*

First the CSI-Experiment process and VeriCoM were analysed through introduction workshops and participation in the experiment in autumn 2017. After gaining insight into the VeriCoM approach, drawbacks of the CSI-experiments conducted without software support, i.e. experiments before autumn 2017, have been identified (see Section 3). These drawbacks showed major shortcomings in the most time consuming parts of VeriCoM, the aggregation and evaluation steps. These were natural candidates for automation through the CSI-Platform. Also the domain model, which evolved over the course of several experiments, needed to be redesigned and enhanced to allow for automation. Furthermore the connection to FigureEight as the crowdsourcing tool needed to be abstracted in order to be able to replace it with a more fitting solution tailored to VeriCoM in the future.

- ***RQ2:*** *What are the requirements for the software platform?*

  Based on the identified drawbacks, functional and non-functional requirements for the CSI-Platform prototype have been defined for each step of VeriCoM (see Chapter 3.6. These requirements were then later addressed during the implementation of the CSI-Platform. While not all requirements were implemented in the prototype they are still valuable starting points for further refinement of the CSI-Platform to adapt it to the needs of future experiments. Requirements elicitation was performed based upon experiences from different experiment rounds, which was satisfying for the CSI-Experiment Spring 2018, but gave only limited insights in the requirements for LEAP (see Section 7.3). An expansion and adjustment of the current requirements to be fitting for LEAP should therefore be considered as future work.

- ***RQ3:*** *How can the CSI domain model be defined in a generic way?*:

  The design of the data model and algorithms showed the importance of thoroughly formalized data model types and concise naming of entities (see Chapter 4). Defect types were introduced to mitigate the insufficient textual description of defect reports, which hampered the application of automated aggregation and evaluation algorithms. Included in this design phase were also, the definition and computation of the evaluation metrics, the determination of defect report correctness and the formalization of the student feedback metrics. During the analysis of VeriCoM, the approach was still under development which lead to various changes in naming and the designed data model as well as the database schema. Also, legacy data from previous CSI-experiments was in an incoherent state, reflecting the evolution of the approach. A lesson to be learned from this is to regularly plan refactoring sessions to make sure that the data model is still suitable for the next experiment phase.

  In the implementation Chapter 5, the developed software prototype for VeriCoM was detailed based upon the abstract definitions presented in the previous design chapter. The implemented classes and their interfaces were described, together with a generic interface to communicate with the crowdsourcing engine FigureEight. This generic definition allows for the implementation of newly developed crowdsourcing engines other than FigureEight. While the developed prototype incorporated

features that reduced work for the experiment administration, it did not offer a finished version of a user interface.

- ***RQ4:*** *Did the developed platform meet the expectations of the experiment administration team?*

  To verify that the developed CSI-Platform met the needs of the CSI-Experiment administration team, the prototype was tested under live conditions during the experiment run of spring 2018. This test-run showed that the CSI-Platform fits the desired specification of a platform to support VeriCoM. Only minor bugs were found and fixed during the first workshop of the experiment. However, as said above, the user interface was not complete and the platform still had to be operated by the developer in order to assure a successful experiment run. This limited the experience of the administration team, whom where not able to work with the CSI-Platform themselves. After the experiment of spring 2018 was conducted, a questionnaire was handed to the administration team to evaluate how satisfied they were with the provided CSI-Platform and to gather further suggestions for improvement. The questionnaire showed that the CSI-Platform was well received and the offered support met the expectations of the experiment team. Due to the limitations stated above, the developed CSI-Platform should only be considered as a prototype. This prototype showed different needs and requirements for the **Le**arning **A**nalytics **P**latform (**LEAP**), which is mentioned as future work in Section 7.3.

As a conclusion to the developed CSI-Platform prototype, it can be said that the designed data model and algorithms as well as the elicited requirements supported the VeriCoM experiments and are a good starting point for the development of future experiment support platforms such as LEAP.

7.2 Limitations

The application of the CSI-Platform during CSI-Experiments is bound to internal and external limitation factors, as well as by the definition of the VeriCoM approach itself. Addressing these limitations helps to understand the methodology of the thesis itself and identifies candidates for future work within this topic.
VeriCoM was applied to a Software Engineering use case as a representative case of verifying conceptual models. The use case focused on the verification of EER model with respect to a textual specification. Thus the CSI-Platform was implemented with this premise. Verifying knowledge graphs from the field of Knowledge Engineering would require an adaptation of the domain model parts which are specific to the current CSI-Experiment setup. Also, the CSI-Platform was tailored to the design of the current crowdsourcing tasks Model Analysis and Defect Validation. While the domain model was designed to be in some way detached from the task design, e.g. by separating judgements from their containing classes, a major change in the overall task structure, would required

an adaptation of the domain model. For the evaluation part of the thesis there is to say that, there was no questionnaire for the students participating in the experiment, which could have identified their satisfaction with the received feedback. This data would be interesting to gather in future iterations. The gathering of experiment data from FigureEight was also not completely automatized and required a manual trigger to start the download of the results. To automatize this step and to move from downloading the whole final results in on batch, to gathering live results during the experiments would improve the overall process. These are some limitations of the current work which should be addressed in future iterations within this experiment family.

7.3 Future work and Learning Analytics Platform (LEAP)

Goal of this thesis was to support CSI-Experiments performed with the VeriCoM-approach through a software platform, the CSI-Platform. While the developed CSI-Platform offered many features to support VeriCoM and reduced the workload for the experiment administration team, it should still be considered a prototype. The core work of the thesis was to formally define the data model and algorithms needed to help automate the VeriCoM steps. This was achieved to a degree that satisfied the experiment management. As was to be expected and intended by the given questionnaire, suggestions for future improvements and the vision for a new Learning Analytics Platform (LEAP) emerged.

The defined data model is sufficient for VeriCoM and extendible for similar use case within this domain, however to cover other Crowdsourced Software Inspection use cases, it still needs to be enhanced and refined to allow for a flexible definition of new data items tailored to these new kinds of inspection approaches. This could include the creation of a user interface to make data model preparation accessible for non-expert experiment team members.

To improve the monitoring of participant(student)-performance, live-feedback during the experiment workshops was mentioned by the experiment team as another desired functionality for the future. Also new performance indicators, rating of participants, new aggregation strategies, filter/sorting options and more evaluation metrics should be developed in the future.

FigureEight was used as a crowdsourcing engine for the course of the conducted experiments. While FE offered sufficient methods to perform the CSI-Experiments for VeriCoM, a more tailored crowdsourcing solution would bring with it greater control over the experiment tasks. Furthermore, replacing the commercial platform FE with a custom solution would reduce the costs for the conduction of the experiments and could be integrated into LEAP.

The vision of LEAP is to provide an online platform for professors, researchers, teachers and students to conduct (crowdsourcing) experiments, university courses and workshops in the fields of Software Inspection, HC&C, Semantic Web and Learning Analytics.

LEAP should be a platform for researchers and students to work together, learn and get feedback on performed experiments and task. It should be adaptable enough to allow for different kinds of experiments and tasks from different university courses. Experiment administrators should benefit from improved experiment/job management, while students receive instant feedback on their performance and their learning progress within each course on a dashboard. With insight on this learning progress, teachers can offer tutorials and tasks to specifically target deficits and support students in their learning outcome. The future development of LEAP would therefore be a great benefit for all those involved and the next step for the CSI-Platform.

List of Figures

1.1 Crowdsourced Software Inspection (CSI) process [29]. 2
1.2 CSI-Experiment setup and underlying process. 4
1.3 Research questions aligned with methods, contributions and chapters. . . 7

2.1 VeriCoM applied to a SE use case [23]. 18

3.1 Guiding questions for the Model Analysis task. 21
3.2 Guiding questions for the Defect Validation task. 22
3.3 The VeriCoM Approach [23]. 22
3.4 Model verification task containing (a) the model element, evidence scenario and model and (b) questions for verification guidance [23]. 23
3.5 Decision tree underlying task design. 25

4.1 VeriCoM approach data model. 36
4.2 Interpretation/Conversion of FigureEight job reports into defect reports. . 46
4.3 Aggregation data model. 48
4.4 Evaluation data model. 50
4.5 Feedback ER-model. 52

5.1 CSI-Platform implementation architecture. 61
5.2 Interpreter-module class diagram. 63
5.3 ModelAnalysisResult interpretation process. 65
5.4 DefectValidationResult interpretation process. 65
5.5 Aggregation-module class diagram. 66
5.6 Aggregation of DefectReports to a FinalDefect. 67
5.7 Evaluation of FinalDefects and TrueDefects to VerifiedDefectReports. . . 67
5.8 Evaluation-module class diagram. 68
5.9 Feedback-module class diagram. 69
5.10 Feedback generation with Workshop- and Worker-statistic. 70
5.11 EME user interface. 72
5.12 DefectReport user interface. 72
5.13 FinalDefect (AggregatedDefectReport) user interface 73
5.14 WorkshopStatistic user interface. 73
5.15 CSI-Experiment results overview. 74

5.16 CSI-Experiment results. 74
5.17 Overview over Workshop "WS1". 75

6.1 CSI-Platform Evaluation method and time-line. 78

List of Tables

2.1 Overview of related work. 17

3.1 Conducted experiments to verify VeriCoM. 20
3.2 VeriCoM 1.: Requirements for the Data Preparation step 31
3.3 VeriCoM 2.: Requirements for the Task Design and Execution step 32
3.4 VeriCoM 3.: Requirements for the Aggregation step 32
3.5 VeriCoM 4.: Requirements for the Evaluation step 33
3.6 5.: Requirements for the Student-Feedback step 33

4.1 EME example 38
4.2 Scenario with evidences for emes example. 39
4.3 TrueDefect example. 41
4.4 TrueDefectNeighbourhood example. 41
4.5 DefectReport example. 42
4.6 AggregatedDefectReport example. 43
4.7 AggregatedDefectReport example. 44

5.1 ModelAnalysisResult csv-record example. 64

6.1 VeriCoM 1.: Data Preparation improvements. 80
6.2 VeriCoM 2.: Task Design and Execution: Download, Interpretation and Conversion of FigureEight results. 80
6.3 VeriCoM 3.: Aggregation improvements 81
6.4 VeriCoM 4.: Evaluation and Experiment/Workshop Statistics improvements 82
6.5 5. Student/Participant feedback improvements. 82
6.6 Answers to S1 - Modelling CSI-Experiment Domain. 84
6.7 Answers to S2 - Data Preparation. 86
6.8 Answers to S3 - Task Design and Execution. 86
6.9 Answers to S4 - Interpretation and Conversion. 87
6.10 Answers to S5 - Aggregation. 88
6.11 Answers to S6 - Evaluation. 89
6.12 Answers to S7 - Student-Feedback. 89

List of Algorithms

4.1 Interpretation of ModelAnalysisJudgements into DefectTypes 47
4.2 DefectType aggregation 49
4.3 DefectValidation-agreement algorithm 56

Bibliography

[1] M. Acosta, A. Zaveri, E. Simperl, D. Kontokostas, F. Flöck, and J. Lehmann. Detecting linked data quality issues via crowdsourcing: A dbpedia study. *Semantic Web*, (Preprint):1–33, 2016.

[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer, 2007.

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

[4] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[5] A. Bernstein, J. M. Leimeister, N. Noy, C. Sarasua, and E. Simperl. Crowdsourcing and the semantic web (dagstuhl seminar 14282). In *Dagstuhl Reports*, volume 4. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[6] M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1:1–182, 2012.

[7] M. Fagan. Design and code inspections to reduce errors in program development. In *Software pioneers*, pages 575–607. Springer, 2002.

[8] T. Gilb, D. Graham, and S. Finzi. *Software inspection*, volume 253. Addison-Wesley Reading, 1993.

[9] J. Howe. Crowdsourcing: A definition. 2006.

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

[11] T. D. LaToza and A. van der Hoek. Crowdsourcing in software engineering: Models, motivations, and challenges. *IEEE software*, 33:74–80, 2016.

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

[13] J. M. Mortensen, E. P. Minty, M. Januszyk, T. E. Sweeney, A. L. Rector, N. F. Noy, and M. A. Musen. Using the wisdom of the crowds to find critical errors in biomedical ontologies: a study of snomed ct. *Journal of the American Medical Informatics Association*, 22(3):640–648, 2014.

[14] J. M. Mortensen, N. Telis, J. J. Hughey, H. Fan-Minogue, K. Van Auken, M. Dumontier, and M. A. Musen. 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, 2016.

[15] M. A. Musen. The protégé project: a look back and a look forward. *AI matters*, 1(4):4–12, 2015.

[16] J. Z. Pan, G. Vetere, J. M. Gomez-Perez, and H. Wu. *Exploiting linked data and knowledge graphs in large organisations*. Springer, 2017.

[17] D. L. Parnas and M. Lawford. The role of inspection in software quality assurance. *IEEE Transactions on Software Engineering*, 29:674–676, 2003.

[18] A. J. Quinn and B. B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1403–1412. ACM, 2011.

[19] K. S. P. Reddy. Introducing jhipster. In *Beginning Spring Boot 2*, pages 279–287. Springer, 2017.

[20] N. B. Ruparelia. Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, 35:8–13, 2010.

[21] G. W. Russell. Experience with inspection in ultralarge-scale development. *IEEE software*, 8(1):25–31, 1991.

[22] M. Sabou and M. Fernandez. Ontology (network) evaluation. In *Ontology engineering in a networked world*, pages 193–212. Springer, 2012.

[23] M. Sabou, D. Winkler, S. Biffl, and P. Penzenstadler. Verifying Conceptual Domain Models with Human Computation: A Case Study in Software Engineering. In *The sixth AAAI Conference on Human Computation and Crowdsourcing*, 2018.

[24] M. Sabou, D. Winkler, and S. Petrovic. Expert sourcing to support the identification of model elements in system descriptions. In *International Conference on Software Quality*, pages 83–99. Springer, 2018.

[25] Y. Sun, A. Singla, T. Q. Yan, A. Krause, and D. Fox. Evaluating task-dependent taxonomies for navigation. In *HCOMP*, pages 229–238, 2016.

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

[27] G. H. Travassos, F. Shull, J. Carver, and V. Basili. Reading techniques for oo design inspections. Technical report, 2002.

[28] L. Von Ahn. Human computation. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 1–2. IEEE Computer Society, 2008.

[29] D. Winkler, M. Sabou, S. Petrovic, G. Carneiro, M. Kalinowski, and S. Biffl. Improving model inspection processes with crowdsourcing: Findings from a controlled experiment. In *European Conference on Software Process Improvement*, pages 125–137. Springer, 2017.

[30] D. Winkler, M. Sabou, S. Petrovic, G. Carneiro, M. Kalinowski, and S. Biffl. Improving model inspection with crowdsourcing. In *Proceedings of the 4th International Workshop on CrowdSourcing in Software Engineering*, pages 30–34. IEEE Press, 2017.

[31] D. Winkler, M. Wimmer, L. Berardinelli, and S. Biffl. Towards model quality assurance for multi-disciplinary engineering. In *Multi-Disciplinary Engineering for Cyber-Physical Production Systems*, pages 433–457. Springer, 2017.

[32] G. Wohlgenannt, M. Sabou, and F. Hanika. Crowd-based ontology engineering with the ucomp protege plugin. *Semantic Web*, 7(4):379–398, 2016.