Interactive scalable condensation of reverse engineered UML class diagrams for software comprehension

Osman, M.H.B.

Part I

Introduction and Background

Chapter 1

Introduction

In this chapter, we present the research context, the problem that we address and the goal of this research. This chapter also provides an overview of the research approach by summarizing the main research steps, the relations between these steps and their purpose. After reading this chapter, the readers should have a high-level understanding of the problem domain, our scoping, and our approach.

1.1 Research Context

Software design is a critical activity in software development. This design embodies the transition from a declarative requirement to a constructive representation that forms the basis for the implementation. Software design is essential to software implementation as well as to software maintenance. Documenting the software design could significantly help the later stages in the software development activity because the design is one of the critical documents to understand the software.

The effort and cost of software maintenance dominate the software development life cycle. The understanding of a software system is one of the most crucial tasks in software maintenance. More than 50% of the time consumed in software maintenance is used for software comprehension. Software documentation, including the software architecture or software design, is a highly useful material for system comprehension. Unfortunately, software documentation is often out-of-sync with the implementation [103],[172]. Reverse engineering is one of the options for recovering software architecture from the implementation code. This method suffers from several problems; one of them is that the resultant diagrams offer too detailed information. Recent Computer Aided Software Engineering (CASE) tools offer to leave out some types of information in software design diagrams (such as class diagrams) by leaving out attributes, operations and parameters. However, these tools are unable to identify the essential

information that helps the developer to understand or focus on specific concerns in system design. In addition, a controlled experiment by Fernandez-Saez et al. [59] found that many subjects did not consider reverse engineered diagrams to be helpful in maintaining software. From their study, they hypothesize that a large amount of data present in the reverse engineered class diagram overwhelms and demotivates users because it surpasses the human capacity for processing information (see e.g. [112]).

The research in this thesis focuses on the software comprehension activity in the software maintenance phase. We aim to provide a method and a tool for software developers to create an overview of their system. In addition, we aim to support the process of understanding software by enabling software developers to create multiple views of their system at various levels of abstraction that may differ for different tasks.

1.2 Problem Statement

When a new software developer is assigned to a maintenance task, several questions commonly arise as the software comprehension activity is started [93][92]. For instance, "Where to start?", "Which classes are important?", "How can I pick up the central classes needed for a more high-level, abstract view which is essential for understanding the model as a whole?" [142] and, "How to make this comprehension task easy?". Because the software documentation is often out-of-sync, these questions are difficult to answer; which makes the software comprehension task more challenging.

As mentioned in the previous section, reverse engineering techniques are capable of recovering a system's structure. A lot of CASE tools provide features that make the reverse engineering process easy to be performed by software developers. However, the resultant class diagrams constructed by these techniques typically contain such a large amount information that it obstructs design comprehension. Building a descriptive and understandable view of the software on the right level of abstraction is one of the most challenging tasks in reverse engineering [162]. This has led us to the following problem statements.

Problem Statement 1. *"How can reverse engineered class diagrams be simplified to assist software understanding?"*

We perceive a need for a framework to condense the reverse engineered class diagrams to improve its understandability. An automatic framework is desired to discover critical information, leaving out unnecessary information, and condense the reverse engineered class diagrams. However, to provide the aforementioned framework, the following issues also need to be addressed.

Problem Statement 2. *"What is the right level of abstraction of class diagrams?"*

We perceive a need for an interactive, scalable condensation of reverse engineered UML class diagram that provides the flexibility to developers to create multiple levels

of class diagram abstraction.

Hence, this research aims to address these issues by devising an automated framework by simplifying UML class diagrams to assist software comprehension. The following subsection explains our research objective to address these issues.

1.3 Research Objective

The central objective of this research is to devise an automated framework for simplifying UML class diagrams to assist the software comprehension task. We use reverse engineered class diagrams (obtained by static analysis) as the primary source of information about a system.

To achieve this, our research focuses on discovering a suitable method to identify the critical and non-critical information in reverse engineered class diagrams. We also aim to provide a prototype implementation of this method through a tool. This prototype should demonstrate the feasibility of the approach. The tool should be interactive and the condensation of class diagrams should be scalable to allow the software developer to generate views of designs at their desired levels of abstraction.

To accomplish our objectives, the following research questions (**RQ**) have been formulated:

Main RQ: *What method of condensing of reverse engineered class diagrams helps developers to understand the design of software systems?*

To answer the **Main RQ**, we need to answer the following RQs:

- **RQ1:** *Which information in class diagrams do developers find important for understanding software designs?*
- **RQ2:** *Which object-oriented design metrics do developers find most indicative for class importance?*
- **RQ3:** *How to automatically condense class diagrams using object-oriented design metrics?*
- **RQ4:** *Can the automatic condensation of class diagrams be enhanced by using class names?*
- **RQ5:** *Does our automated framework for condensing class diagrams help developers to understand the design of software systems?*

1.4 Research Methods

The main objective of this research is to discover a method of enhancing the comprehension of reverse engineered class diagrams. To accomplish this goal, we apply various research methods, including: surveys [137], case studies [46] and experiments [148]. Details about the research methods used are provided in Table 1.1.

Table 1.1: *Research Methods used in this Research*

Chapter	Methodology	Primary Objective	Primary Data
3	Field Study	Descriptive	Qualitative
4	Experiment	Descriptive	Quantitative
5	Survey	Descriptive	Qualitative
6 & 10	Survey	Descriptive	Quantitative
7 & 8	Experiment	Validation	Qualitative

In summary, we used surveys for eliciting information on how the software engineers think classes diagrams could be simplified. An experiment is used to explore the state-of-the-art of reverse engineering class diagrams. A field study [185] is used to explore the usage of UML diagrams in open source software development. We used experiments to explore and validate the effectiveness of some class condensation techniques that we developed. We also used the survey method to validate our proposed automated framework for condensing class diagrams.

We provide our experiments' material (online) for the purpose of external replication and future research ([6] [122] [134]).

1.5 Roadmap

This section presents an outline (see Figure 1.1) of the chapters in this thesis. We summarize the purposes of each chapter and relate the chapters to the research questions. Also, we relate the chapters to our publications.

- **Chapter 2: Definition.** The purpose of this chapter is to define the principal concepts used in this research. We briefly describe the Unified Modeling Language (UML) and class diagrams. Also, we describe the concepts of forward and reverse engineering, the basics of machine learning and explain the notion of software comprehension.

- **Chapter 3: UML Usage in Open Source Software Development.** The purpose of this chapter is two-fold: i) To present the examples on the use of UML diagrams in Open Source Software Development (OSSD) and ii) To find suitable case studies for automatic condensation of class diagrams research. For this purpose, we select ten OSSD projects from different types of domains. We assess the UML usage of OSSD projects, the level of detail (LoD) and the frequency of updating diagrams. Our findings also cover the application of UML modeling in different level of detail for different purposes, a change in focus on types of diagram used over time, and findings on how the size of models relates to the size of the implementation.

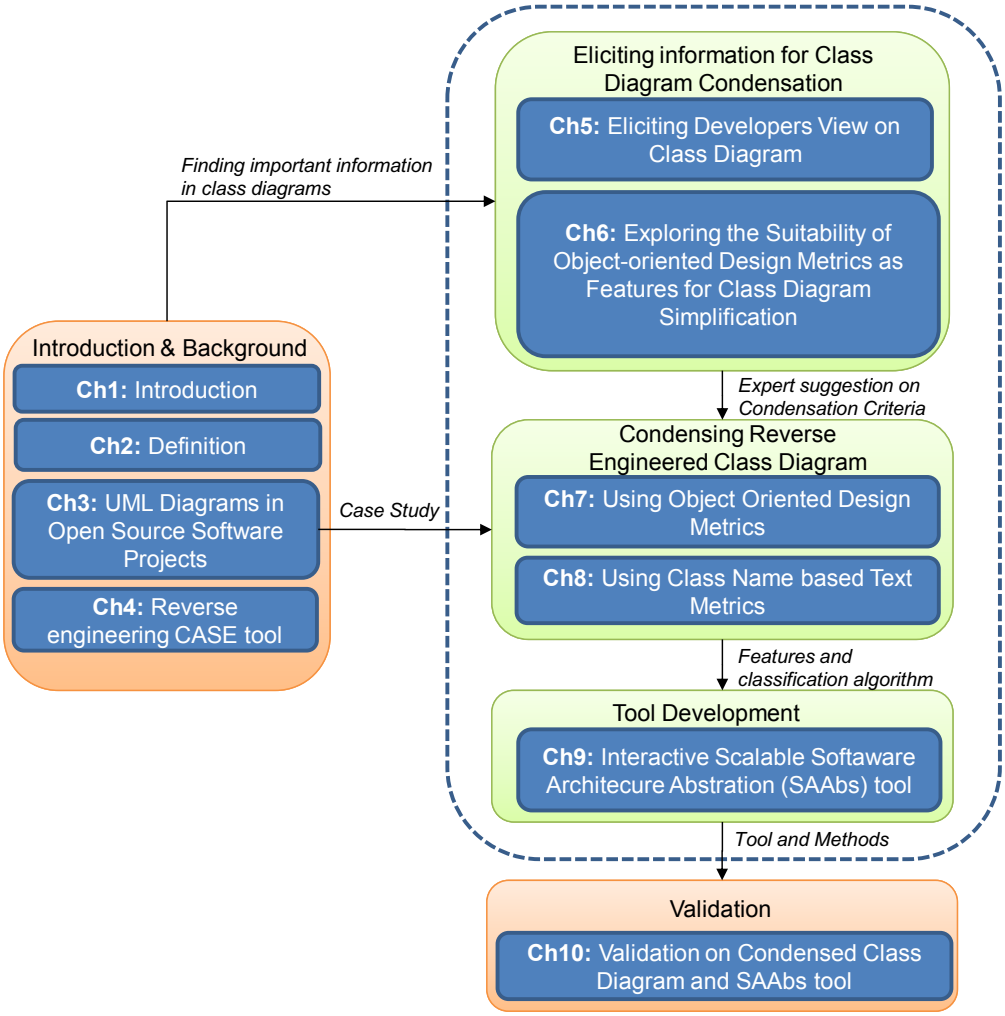

Figure 1.1: *Thesis Roadmap*

This chapter is a more detailed version of the following publication:

- Hafeez Osman and Michel R.V. Chaudron (2013). **UML Usage in Open Source Software Development : A Field Study.** *In Proceedings of the 3rd International Workshop on Experience and Empirical Studies in Software Modelling (EESSMod 2013), pages 23-32, Miami, USA*

- **Chapter 4: Assessing the Correctness and Completeness of UML CASE tools in Reverse Engineering.** The main purpose of this chapter is to demonstrate the state-of-the-art of CASE tools for reverse engineering of source code into

class diagrams. We assess the strengths and the weaknesses of the reverse engineered class diagrams constructed by eight common CASE tools. We compare and evaluate the types of input, the types of reverse engineered diagrams that could be constructed, and the quality of resulting diagrams. This chapter covers information about the correctness, completeness and the quality of the reverse engineered class diagrams (as constructed by CASE tools). The results provide a baseline of current reverse engineering of class diagrams by CASE tools.

This chapter is adapted from the following publications:

- Hafeez Osman and Michel R.V. Chaudron (2011). **An Assessment of Reverse Engineering Capabilities of UML Case Tools.** *In Proceedings of the 2nd Annual International Conference on Software Engineering and Applications (SEA 2011), pages 7-12, Singapore*
- Hafeez Osman and Michel R.V. Chaudron (2012). **Correctness and Completeness of CASE tools in Reverse Engineering Source Code into UML Model.** *GSTF Journal on Computing vol.2, num.1, pages 193-201*

- **Chapter 5: Eliciting Developer's Views on Simplifying Class Diagrams.** In this chapter, we aim to discover how to simplify class diagrams in such way that the system is easier to understand. For this purpose, we conduct a semi-structured survey to gain knowledge about the criteria that developers believe are relevant for including or excluding in class diagrams. The results of this survey suggest what are the important elements in a class diagram.

  This chapter answers **RQ1** and it is a more detailed version of the following publication:

  - Hafeez Osman, Arjan van Zadelhoff, Dave R. Stikkolorum and Michel R.V. Chaudron (2012). **UML Class Diagram Simplification: What is in the Developer's Mind?** *In Proceedings of the 2nd International Workshop on Experience and Empirical Studies in Software Modelling (EESSMod 2012), pages 31-36, Innsbruck, Austria*

- **Chapter 6: Exploring the Suitability of Object-oriented Design Metrics as Features for Class Diagram Simplification.** The purpose of this chapter is to identify suitable design metrics that influence the determination of class inclusion and exclusion. We conduct a survey to investigate the suitability of object-oriented design metrics (from software documents) in deciding on the inclusion and exclusion of classes from class diagrams. The results indicate what software design metrics are most important to users to decide whether to include a class in a class diagram.

  This chapter answers **RQ2**. It is a more detailed version of the following publication:

  - Hafeez Osman, Arjan van Zadelhoff and Michel R.V. Chaudron (2012). **UML Class Diagram Simplification - A Survey for Improving Reverse Engineered Class Diagram Comprehension.** *In Proceedings of the 1st International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2013), pages 291-296, Barcelona, Spain*

- **Chapter 7: Condensing Reverse Engineering Class Diagram using Object-Oriented Design Metrics.** The purpose of the study in this chapter is to investigate the usefulness of object-oriented design metrics as features for identifying class inclusion and exclusion. To this end, we conduct an experiment for condensing reverse engineered class diagrams on the basis of software design metrics. We use object-oriented design metrics as features for applying machine learning classification approaches to classifying the classes for inclusion and exclusion. The machine learning is used because it provides an automated method for the classification process. Nine OSSD projects are used as case studies. This experiment also identifies the classification algorithms that perform best for this purpose.

  This chapter answers **RQ3** and it is adapted from the following publication:

  - Hafeez Osman, Michel R.V. Chaudron and Peter van der Putten (2013). **An Analysis of Machine Learning Algorithms for Condensing Reverse Engineered Class Diagrams.** *In Proceedings of the 29th International Conference on Software Maintenance (ICSM 2013), Eindhoven, the Netherlands*

- **Chapter 8: Condensing Reverse Engineered Class Diagrams through Class Name Based Abstraction.** The purpose of this chapter is to improve the classification of class inclusion and exclusion by using class names. We formulate text metrics based on the frequency of occurrence of words in class names. We explore multiple combinations of features and compare the results with the previous outcomes (Chapter 7). The evaluation is performed using 10 OSSD projects. This chapter presents the improvement of class inclusion and exclusion classification, by using text and object-oriented design metrics as features.

  This chapter answers **RQ4** and it is adapted from the following publication:

  - Hafeez Osman, Michel R.V. Chaudron and Peter van der Putten (2014). **Condensing Reverse Engineered Class Diagrams through Class Name Based Abstraction.** *In Proceedings of the 2014 World Congress on Information and Communication Technologies (WICT), Malacca, Malaysia*

- **Chapter 9: Interactive Scalable Abstraction of Reverse Engineered UML Class Diagrams.** In this chapter, we demonstrate our automated Software Architecture Abstraction (SAAbs) framework for simplifying class diagrams based on class inclusion/exclusion (Chapter 7 and 8). The SAAbs framework applies a machine

learning classification algorithm to produce a class importance ranking for all classes in a reverse engineered class diagram. This ranking is used for scalable abstraction and visualization of the class structure of the system. We created a tool that allows developers to interactively explore a reverse engineered class diagram at multiple levels of abstraction.

Part of this chapter is adapted from the following publication:

- Hafeez Osman, Michel R.V. Chaudron and Peter van der Putten (2014). **Interactive Scalable Abstraction of Reverse Engineered UML Class Diagrams.** *In Proceedings of the 21st Asia-Pacific Software Engineering Conference (APSEC 2014), Jeju, Korea*

- **Chapter 10: Validation.** In this chapter, we conduct a user study to validate the SAAbs framework and tool in providing a platform for assisting developers to comprehend reverse engineered class diagrams. This chapter aims at i) discovering the understandability of condensed class diagrams, ii) finding whether the condensed class diagram generated by this approach is helpful in understanding the software design and, iii) eliciting the usefulness of the SAAbs tool in assisting software developers to understand the software.

  This chapter answers **RQ5**. Part of this chapter is adapted from the following publication:

  - Hafeez Osman, Michel R.V. Chaudron and Peter van der Putten (2014). **Interactive Scalable Abstraction of Reverse Engineered UML Class Diagrams.** *In Proceedings of the 21st Asia-Pacific Software Engineering Conference (APSEC 2014), Jeju, Korea*

- **Chapter 11: Conclusion.** In this chapter, we summarize the results, draw conclusions and discuss future work.