Thesis for The Degree of Licentiate of Engineering

Understanding Software Design for Creating Better Design Environments

Rodi Jolak

Division of Software Engineering
Department of Computer Science & Engineering
Chalmers University of Technology and Göteborg University
Göteborg, Sweden, 2017

Understanding Software Design for Creating Better Design Environments

Rodi Jolak

Technical Report No 168L
ISSN 1652-876X
Department of Computer Science & Engineering
Division of Software Engineering
Chalmers University of Technology and Göteborg University
Göteborg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Göteborg, Sweden 2017.

"There is no unique picture of reality"
- Stephen Hawking

Abstract

Context: Software design is considered an essential activity to analyze software requirements in order to produce a description of the software's internal structure that will serve as the basis for its construction. Models are a means to describe complex systems at several levels of abstraction and from a diversity of perspectives. Surprisingly, most of the current software design environments are not based on understanding of real needs of software designers in practice.

Objective: As a first step towards supporting realistic software design processes, this thesis focuses on understanding software design practices, as well as on proposing and assessing of a new generation of software design environments.

Method: To achieve the objective of this research, design science and empirical methods are employed. In particular, a new generation software design environment, called *OctoUML*, is created. Furthermore, to understand whether there is a need to improve modeling tools, the modeling process is analyzed in order to reveal how much effort is given to designing (i.e. thinking about the design of software systems), and how much effort is given to drawing the model (i.e. tool interaction).

Result: This thesis describes two areas of contributions: On the one hand, *OctoUML* is perceived a usable environment in terms of ease of use, efficiency and user satisfaction. Moreover, it seems that *OctoUML* supports the design process by bridging the gap between early-phase design process and later on documentation and formalization process. Further results show that *OctoUML* was not only enjoyed by the designers, but also enhanced the efficiency of the software design process. On the other hand, we proposed experiments to increase our understanding of the software design process. We elicit many issues that need to be considered in such experiments. Our initial findings suggest that the majority of the modeling effort is devoted on design thinking. However, the effort spent on using modeling tools should be reduced by investigating better modeling-tool support.

Keywords: Software Engineering, Software Modeling, Design Effort, Software Design Environments, UML, Empirical Software Engineering

Acknowledgment

I would like to express my sincere gratitude to a number of people who helped me in accomplishing this research:

To my main supervisor Michel R.V. Chaudron for his continuous support, inspiring discussions and precious suggestions. To my co-supervisors: Morten Fjeld and Eric Knauss for their collaboration, tips, feedback and interesting discussions. To my examiner Ulf Assarsson for his constructive feedback during the follow-up meetings. To professor Matthias Book for accepting to be the discussant of my licentiate seminar.

To Arif Nurwidyantoro, Boban Vesin, Eric Umuhoza (the great *ninja*), Marco Brambilla, Marcus Isaksson, Truong Ho-Quang for their contributions to the papers appended in this licentiate thesis.

To all my colleagues (researchers, administrative officers and coordinators) at the software engineering division for their support and help. Special thanks to all the PhD students. In particular, I would like to thank my office-mate Truong for his support, funny discussions, gym & ping-pong sessions and also for ruining my flight to MODLES 2016. Special thanks to Grischa Liebel for reviewing the *kappa*. Furthermore, I would like to thank: Salome Maro, Hugo Sica de Andrade and Federico Giaimo for being such nice neighbours, and Hiva Alahyari for showing me that *Soranî* is not too much different from *Kurmancî*.

To Bilal karasneh, Alexandru Dancu, Duy Le Khanh and Dave Stikkolorum for their feedback and interesting discussions regarding *OctoUML*.

To my parents for motivating me to work hard and persevere to achieve my goals. To my sisters and finacée for their love, constant support and continuous encouragement.

List of Publications

Included publications

This thesis is based on the following publications:

[A] M.R.V. Chaudron, R. Jolak "A Vision on a New Generation of Software Design Environments"
In First International Workshop on Human Factors in Modeling (HuFaMo 2015). CEUR-WS, pp. 11-16. 2015.

[B] R. Jolak, B. Vesin, M. Isaksson, M.R.V. Chaudron "Towards a New Generation of Software Design Environments: Supporting the Use of Informal and Formal Notations with OctoUML"
In Second International Workshop on Human Factors in Modeling (HuFaMo 2016). CEUR-WS, pp. 3-10. 2016.

[C] R. Jolak, B. Vesin, M.R.V. Chaudron "Using Voice Commands for UML Modelling Support on Interactive Whiteboards: Insights and Experiences"
In Proceedings of the 20th Ibero American Conference on Software Engineering (CibSE) @ICSE17, pp. in print. 2017.

[D] R. Jolak, B. Vesin, M.R.V. Chaudron "OctoUML: An Environment for Exploratory and Collaborative Software Design"
In Proceedings of the 39th International Conference on Software Engineering, pp. 7-10. 2017.

[E] R. Jolak, E. Umuhoza, T. Ho-Quang, M.R.V. Chaudron, M. Brambilla "Dissecting Design Effort and Drawing Effort in UML Modeling"
In Proceedings of 43th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. in print. 2017.

Other publications

The following publications were published during my PhD studies, or are currently in submission. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[a] B. Karasneh, R. Jolak, M.R.V. Chaudron "Using Examples for Teaching Software Design: An Experiment Using a Repository of UML Class Diagrams"
2015 Asia-Pacific Software Engineering Conference (APSEC). IEEE, pp. 261-268. 2015.

[b] R. Jolak, E. Umuhoza, M.R.V. Chaudron, M. Brambilla, A.Pierantonio "Analyzing the Effort of Software Modeling"
In Submission to the Third International Workshop on Human Factors in Modeling (HuFaMo 2017). CEUR-WS, 2017.

Research Contribution

My contributions to Paper A are mainly based on identifying challenges in software design processes and consulting the related work. I also contributed in writing the vision on new generation software design environments.

The design decisions of *OctoUML* were performed by Michel R.V. Chaudron and me. The first version of *OctoUML* was mainly created by Marcus Isaksson and Christophe Van Baalen. Between whiles, I also contributed in the development of different functionalities of *OctoUML*. The voice recognition component was added into *OctoUML* by Johan Hermansson and Emil Sundklev, under my supervision and assistance.

In Paper B and Paper C, I did the majority of the paper writing. In particular, my contributions are the related work, approach, user studies preparation, results analysis and discussion.

In Paper D, the demonstration video was done in collaboration with Boban Vesin. The main contributions of this paper are the demo video and the discussion of the evaluations done in Papers B and C. Boban Vesin and I did the majority of paper writing.

Finally in Paper E, the experiments were conducted in collaboration with *Polytechnic University of Milan* and *Gadjah Mada University*. My major efforts were experiment design, effort estimation approach, data analysis and results discussion. In terms of paper writing, I was the major author of all sections, but the experiment definition and execution details.

Contents

Chapter 1

Introduction

In this chapter, we discuss the context of the thesis, the research motivation, research goals and research questions. Also, we outline the research methods and research contributions. Finally, we conclude and discuss future directions.

Models are useful means to comprehend large and complex software systems. In fact, models increase the abstraction level and consequently provide a simplified representation of a system [1]. Software models provide means for expressing software designs. Software designers often use software modeling tools to create software designs.

We classify modeling tools into two categories: informal and formal. On the one hand, we mean by informal tools any tool that supports informal modeling in the sense that it does not constrain the modeling notations that can be used by software designers. Examples of informal tools are *whiteboards* and *pen & paper* (see Figure 1.1). These tools are flexible, easy to use and allow designers to unleash their expressiveness. On the other hand, we mean by formal tools any Computer Aided Software Engineering (CASE) tool (see Figure 1.1) that supports the creation of designs expressed through formal modeling notations.

Figure 1.2 presents different purposes of using informal and formal modeling. The main purpose of using informal tools is their simplicity and flexibility. In particular, these tools promote ideation, problem exploration and understanding, communication and creative thinking [2]. The main purpose of using formal tools is documentation and code generation [3].

Current informal tools do not serve well for persistence and transfer of designs. What typically happens in practice is that software designers go to the whiteboard, create a software design, take a picture of the created design, go back to their desks, run a CASE tool, and try to re-draw or formalize the design that they have previously created on the whiteboard. In this process, redrawing is an additional step using a separate tool.

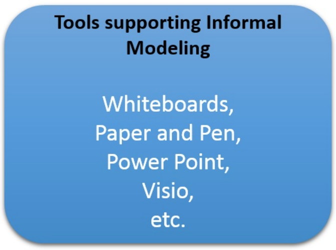


Figure 1.1: Tools supporting informal and formal software modeling

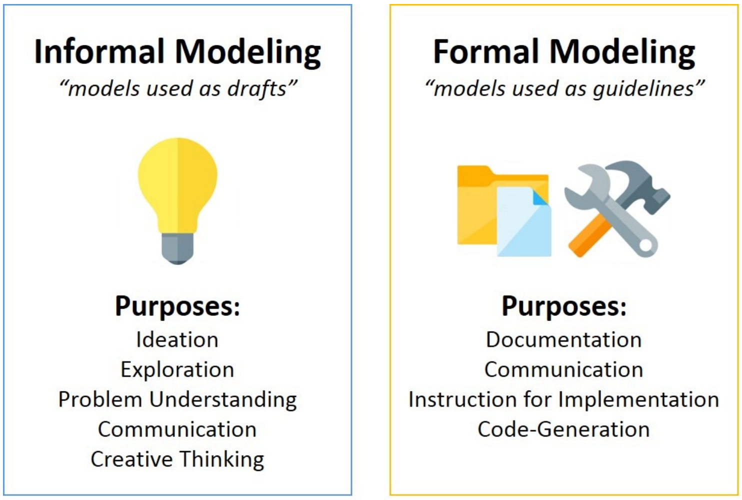


Figure 1.2: The purposes of using informal and formal modeling

A common weakness in formal tools is poor support of realistic design practices [4]. Indeed, they support one or few formal modeling notations, and hence restrict designers' expressiveness. To make matters worse, the majority of CASE tools are not designed for user experience, and their usability leaves to be desired [5]. In fact, there are reports showing that CASE tools are perceived complex and difficult to use [6, 7]. Moreover, the complexity of CASE tools is considered to be a reason that adversely affect the adoption of model-based approaches [8].

Both informal and formal modeling have their advantages and disadvantages. Yet, neither serves all purposes in software designing. Therefore, in this thesis we study new forms of software design tools which could better serve the multitude of purposes.

In the next section, we provide more details regarding the aforementioned issues. In particular, we give an overview of existing software design challenges and describe the motivation of this research.

The rest of the this Chapter is structured as follows: The research motivation and focus are presented in Sections 1.1 and 1.2, respectively. Section 1.3 describes the background on software modeling and design 1.3.1, as well as on software design environments 1.3.2. The related work are described in Section 1.4. The methodology of the research is described in Section 1.5. Section 1.6 reports the contributions of this thesis. Finally, the conclusion is stated in Section 1.7, and the future work is illustrated in Section 1.8.

1.1 Research Motivation

In this section, we describe existing software design challenges that motivate this research. Our aim is to find solutions to accommodate these challenges.

C1. Support of Informal and Formal Modeling. Software designers often go to the whiteboard to discuss requirements, explore domain problems and sketch design solutions. This is because whiteboards are flexible mediums and at easy disposal. Furthermore, whiteboards allow informal modeling i.e. there are no restrictions on the type or formality of the modeling notations that can be used. However, whiteboards do not offer means for data processing and persistence. Thus, the sketched diagrams often need to be formalized and re-modeled again using a CASE tool. The transition between informal and formal tools introduces a discontinuity that can be a source of errors. Indeed, rationale, ideas and the logical basis for design solutions can be easily lost when moving from the whiteboard to the CASE tool [4]. Moreover, CASE tools provide environments where only one or few formal modeling notations are supported. This may actually limit the expressiveness of designers, especially in early-phase software design. Table 1.1 is based on [9], and describes some advantages of informal and formal modeling tools.

The main challenge is to provide a *'one stop'* environment capable of supporting both informal and formal modeling, while preserving the advantages of both informal and formal tools.

C2. Usability. CASE tools are criticized of being complex and difficult to use [6,8]. The Complexity of CASE tools often costs companies extra money for training and learning endeavor. In particular, the interaction with these tools is not always well-designed for a user experience, easy learning and effective use [8]. As a consequence, CASE tools are often considered as barriers to the adoption of model-based approaches [11]. The challenges are to

Table 1.1: Advantages of informal and formal modeling tools

	Informal Tools	**Formal Tools**
Clarity		High clarity because of strict adherence to syntax
Flexibility	Caters for improvisation of notation	
Ease of continuous design	In tools based on digital editing, editing (move, resize, delete, undo, etc.) is easier than in sketch-based tools such as whiteboards.	
Ease of learning Notation		Formal syntax checking helps in learning the proper syntax
Intuitiveness of using tool	Very simple to use; but limited in functionality	More difficult to learn, but advanced functionalities supported
Collaboration	Multiple people collaborating on a shared design prefer to use informal representations [10]	
Integration	Absence of a formal syntax (and semantics) prohibits exchange of designs	Formal syntax allows a formal representation of the design that can be exchanged with other tools

- provide, as well as combine rich features in a simple and intuitive User Interface (UI), and
- make CASE tools fit easily into users' activities, rather than forcing users to fit their activities into the dictates of the tools [12].

C3. Interaction Modalities. One key aspect of usability is the manner in which people interact with the system. The interaction with current CASE-tools takes place essentially by using the mouse and keyboard. However, in recent times interaction technologies such as touch, voice and gesture [13] have matured, and become commonly used as interaction modalities with software systems. Such interaction modalities shift the balance of human-computer interaction much closer to the human. This leads to the following challenges in the area of software design tools:

- to open up new cognitive dimensions , and
- to provide more intuitive and effective interaction with CASE tools.

C4. Collaboration. In practice, typically multiple engineers work together on creating a design. CASE tools are often deployed on personal computers, and only one designer can effectively interact with the PC at any one time. This actually limits the collaboration between designers. Furthermore, in some situations designers are geographically distributed and need tool-support to accommodate remote-collaborative design sessions [14]. The challenges are to:

- provide efficient support for collocated software design sessions, and

- accommodate distributed software design, while preserving the natural, effortless kind of awareness and communication that happens in collocated settings.

C5. Modeling Effort. Regardless of the reported benefits of model-based approaches in literature, e.g. [15, 16], the use of modeling is still debated in practice. This is because modeling is believed as an unnecessary and superfluous activity that introduces an extra-effort in the process of software development [17]. Part of the modeling effort is actually devoted to reasoning and thinking about the design solution, while other parts of the effort are dedicated to drawing the design solution (i.e. representing the solution via a modeling notation). Here the challenge is to increase our understanding of the amount of effort spent on *design thinking* and *model drawing*, as well as on how to reduce the drawing effort (i.e. the modeling cost).

1.2 Research Focus

In this section, the research goals are described in detail. Afterwards, the research questions are presented.

1.2.1 Research Goals

This research is motivated to address the aforementioned challenges in Section 1.1. The overall goal of the PhD research is "*to support a realistic software design process by understanding software design practices, as well as proposing and assessing a new generation of software design environments*".

As first steps towards achieving the overall research goal, the following goals are addressed in this licentiate thesis:

- **Goal 1:** To better understand the process of software design, in particular in relation to different design-purposes,
- **Goal 2:** to propose a vision on a new generation of software design environments,
- **Goal 3:** to create a proof of the proposed concept, and
- **Goal 4:** to assess the created concept, as well as evaluate the efficiency and usability thereof.

To achieve *Goal 1*, we study and analyze the process of software design. We investigate existing software design challenges and study modeling tool-support. In practice, software modeling is criticized of being a time-consuming, excessive and unnecessary approach [18]. Indeed, one argument in the discussion about the adoption of model-based approaches in industry is the supposedly large effort it takes to do modeling. As a first step to achieve *Goal 1*, we observe the modeling process and reveal how much effort is given to *design* (i.e. thinking about the design of software systems), and how much effort is given to *drawing* the model (i.e. tool interaction). Based on our findings, we explore whether it is worthy to do modeling, and whether there is a need to investigate better tool-support.

For *Goal 2*, we present a vision for a more intuitive, inspiring and efficient software design environments to support exploratory and collaborative software modeling. In particular, we describe innovative ideas and novel concepts which we consider relevant in supporting the software design process.

To realize the vision of the conceptual solution, a new generation software design environment (called *OctoUML*) is created (*Goal 3*). *OctoUML* can be deployed on a number of input devices ranging from desktop computers over large touch screens to large interactive whiteboards. The main objective of *OctoUML* is to build a bridge between informal modeling and formal modeling

CASE tools are frequently considered complex and time-consuming [6–8]. Software design environments should provide efficient support for designing software system. For this, we perform an evaluation of our created concept by assessing its efficiency and usability (*Goal 4*). Furthermore, Cohen and Oviatt [13] illustrated the importance of systems that support multiple modes of interactions in reshaping daily computing tasks. For this, we investigate experiences from enabling new interaction modalities within our created concept.

1.2.2 Research Questions

To reach the goals of this licentiate thesis, the following research questions are formulated:

- **RQ1.** How can a software design environment integrating the power of both formal and informal tools be achieved?
- **RQ2.** How can modeling tools be improved to be easier to use and more productive?
- **RQ3.** What are the perceptions regarding the usability of the proposed conceptual solution, *OctoUML*?
- **RQ4.** Does tool-support for mixing informal and formal notation better support the software design process?
- **RQ5.** Could the employment of voice interaction modality within the software design environments enhance the efficiency of the software design process?

Table 1.2: The research questions and the targeted research goals

Licentiate Goal	G1	G2	G3	G4
Description	To better understand the process of software design	Propose a vision on a new generation of software design environments	Create a proof of the proposed concept	Assess the created concept
Research Questions	RQ6, RQ6.1 & RQ6.2	RQ1 & RQ2	RQ1 & RQ2	RQ3, RQ4 & RQ5

- **RQ6.** Can we dissect the design effort and drawing effort in software modeling?

  RQ6.1. How much of the modeling effort is spent on design?

  RQ6.2. How much of the modeling effort is spent on drawing the design solution?

Table 1.2 reports the research questions and the targeted research goals. In particular, **RQ1** and **RQ2** aim to achieve **Goal 2** and **Goal 3**. **RQ3**, **RQ4** and **RQ5** target **Goal 4**. Finally **RQ6** and its sub-questions **RQ6.1** and **RQ6.2** address **Goal 1**.

1.3 Background

In the first part of this section, we present the concepts of software design and modeling. In particular, we give an overview about the characteristics of these two activities. Later on, we provide details about existing modeling tools and their types.

1.3.1 Software Design and Modeling

Design. According to Ralph and Wand [19], design is "*a **specification** of an **object**, manifested by some **agent**, intended to accomplish **goals**, in a particular **environment**, using a set of **primitive components**, satisfying a set of **requirements**, subject to some **constraints***".

Their conceptual model of design is presented in Figure 1.3. The design specification is a detailed description of the structure of an object or entity that is being developed. The design agent is the entity that specifies the structural properties of the design object. The environment is the context in which the design object is intended to exist. Goals are what the design object should achieve, while the primitives are the elements that compose the object. The requirements are set of structural (unresponsive to environment changes or stimuli) or behavioural (responsive to environment changes) characteristics

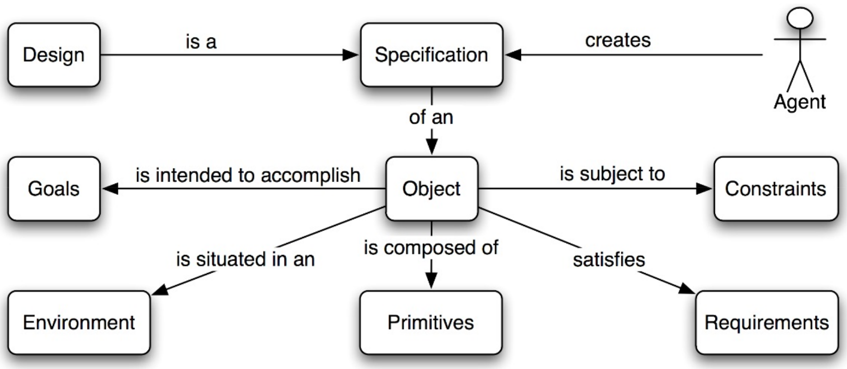


Figure 1.3: Conceptual model of "Design"

that the design object should have, whereas the constrains could be structural or behavioural restrictions on the design object.

The nature of software design. Bourque et al. [20] define software design as "*the software engineering life cycle activity in which software requirements are analyzed in order to produce a description of the software's internal structure that will serve as the basis for its construction*".

The purpose of the design is to produce a solution to a problem [21]. The problem is basically described by means of the requirements specification, and the solution is given by the designer via describing how the requirements can be addressed. Therefore, design is a problem-solving task which asks designers to consider and evaluate different design decisions (i.e. considering size, performance, complexity, etc.). Given that the design process lacks any analytical form, there might be many acceptable solutions to the same design problem. This gives the design problem a *wicked nature*. In a wicked problem, designers do not have a clear understanding of what can be the final solution and have only a vague goal in their mind in the beginning [22]. A solution of one aspect of a wicked problem simply changes the problem, and may also pose more problems.

In problem-solving, *abstraction* is useful to separate the logical and physical aspects of the design of a system. Abstraction has a central role in design. It is a technique for dealing with complexity of software systems. Furthermore, it enables to consider the problem by removing low-level details from the description and retaining the essential properties. Kramer [23] stated that abstraction is especially essential in solving complex problems, as it allows the problem solver to think in terms of conceptual ideas rather than in terms of their details.

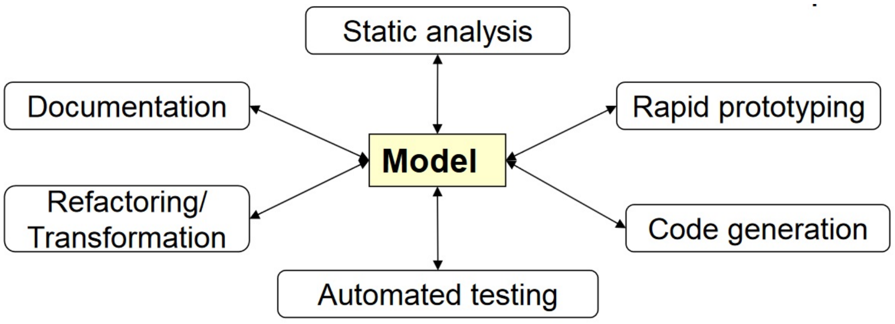


Figure 1.4: In MDE, models are the main artifacts of software development

Software Modeling. In practice, modeling and design go hand-in hand. A model is a simplified or partial representation of reality, defined in order to accomplish a task or to reach an agreement. Software models provide means for expressing a software design. Indeed, they allow to describe, reason, predict and evaluate both software problems and solutions based on different levels of abstraction and from multiple perspectives [1]. Model Driven (MD) approaches are introduced in software engineering since many years. These approaches are increasingly considered promising for large or critical systems as a solution to issues of large, complex or critical software systems. Modeling languages can be General Purpose Language (GPL) or Domain Specific Language (DSL). GPLs can be applied to any domain. Examples of GPLs are: Petri nets [24] and the Unified Modeling Language (UML[1]). DSLs are designed specifically for a certain application domain or context. Examples of DSLs are: HTML[2] and SQL [25].

There are many model-driven approaches adopted in practice. To mention some of them:

- Model Based Engineering (MBE): exploits models in (document-centric) activities. Models play an important but not the primary role [26].
- Model Driven Engineering (MDE): is a model-centric paradigm where models are the main artifacts of software development [27]. Documents such as code and test cases can be obtained automatically (see Figure 1.4). The main benefits of MDE are to improve productivity and quality, as well as support early defect detection and maintenance.
- Model Driven Development (MDD): is a development paradigm that

[1]http://www.omg.org/spec/UML
[2]https://www.w3.org/html/

uses models as the primary artifact of the development process [28]. The idea is that through the use of diagrams, systems can be specified to a modeling tool and then the code is generated in a conventional programming language.

- Model Driven Architecture (MDA): is the particular vision of MDD proposed by the Object Management Group (OMG[3]).

MBE has been applied effectively in several application sectors, e.g. embedded systems [29] and telecommunication [30]. Indeed, most increments in software productivity are obtained by increasing the abstraction level.

1.3.2 Software Modeling Tools

Software designers often use software modeling tools to perform a software design. Design support tools can be divided in two categories: informal tools and formal tools.

Informal tools support the creation of informal designs (i.e. elements and symbols that do not adhere to a modeling language or syntax). Such tools are typically used for problem domain exploration, ideation, creativity and solution discussion. A typical example of an informal tool is the whiteboard (see Figure 1.5). Whiteboards are flexible mediums and do not constraint the notation that can be used.

Formal tools enforce the use of formally defined syntax of some modeling languages. Indeed, they typically provide support for the creation of one or more formal notations that are adherent to one or more modeling languages, such as the UML. Typical examples are CASE tools (like Rational Rose, Enterprise Architect, Visual Paradigm, StarUML etc.). These tools are mainly used to describe the software system, as well as to provide instructions for software implementation.

1.4 Related Work

In this section we describe some related work that focus on understanding and supporting software design.

1.4.1 Understanding Software Design

Petre and Van Der Hoek [31] organized a workshop to study professional software designers in action. They provided video-recordings and transcripts of three two-person teams who were assigned to create a software design for the

[3]http://www.omg.org/

Figure 1.5: Informal modeling on the whiteboard

same set of requirements on a whiteboard. Over fifty researchers were invited to participate in the workshop. They were asked to explore the videos to understand design strategies and types of activities that professional designers engage in during software design sessions. In the following paragraphs, we report the findings of some of the researches who took a part in the workshop.

Tang et al. [32] tried to understand how software designers make design reasoning and decisions, and how the decision making process influences the effectiveness of the design process. They found that planning design discussions in an opportune way improves the exploration of design solutions. They also found that the manners by which the decisions are made have an impact on the use of design time and derived solutions. Furthermore, it seems that the use of design reasoning techniques contribute to the effectiveness of software design.

Sharif et al. [33] analyzed the video-recordings and explored the software design strategies and activities that happened in each design session. They identified some of time-consuming design activities such as decisions about the logic, discussion of uses cases, drawing class diagrams, and drawing the user interface. Furthermore, they found that planning and high degree of agreement between designers play a significant role in delivering a detailed design that covers most of the requirements.

Baker and Van Der Hoek [34] studied the video recordings in order to understand how software designers address software design problems. In particular, they evaluated the design sessions in terms of the discussed subjects, generated ideas and design cycles. They found that the design sessions were highly incremental and designers repeatedly returned to high level subjects, as well as to previously discussed ideas.

Budgen [4] tried to observe the design sessions and identify principles for designing software deign environments. He found that: (i) informal diagrams and notations are often used, (ii) there was a frequent switching between viewpoints and sections of the whiteboard, and (iii) the whiteboard annotations were chiefly performed by one person.

1.4.2 Supporting Software Design

Based on his observations that are reported previously, Budgen [4] provided some recommendations for future design support tools. In particular, informal diagrams and lists need to be integrated with other more formal notations. Ideally, a tool should be simple and support the transition from informal notations to formal notations. He also stated that unless it transformed into more formal description during early-phase design, much of reasoning and rationale would quickly be lost.

Cherubini et al. [2] highlighted how the software design on whiteboards is prevalent and transient, serving a crucial role in supporting the development process. They stated that whiteboards are preferred for design because of their flexibility and simplicity.

Derkel and Herbsleb [35] observed collocated object-oriented design collaborations and focused on representation use. They found that teams intentionally improvise representations and sketch informal diagrams in varied representations that often diverge from formal notations or languages. To support collaborative software design, they suggest that collaborative software design environments should focus on preserving contextual information, while allowing unconstrained mixing and improvising of notations.

Whittle et al. [8] explored tool-related issues in affecting the adoption of MDE in industry. Based on interviews conducted with twenty companies, they observed that the interviewees emphasized tool immaturity, complexity, and lack of usability as major barriers against the adoption of MDE. Usability issues were reported to be related to the complexity of user interfaces. There is a lack of consideration of how people work and think. The authors suggested to match modeling tools to people, not the other way around, by producing more useful and usable tools, as well as supporting early-phase design and creativity in modeling.

1.5 Research Methodology

To achieve the research goals of this thesis, design science and empirical methods are employed.

1.5.1 Design Science

Design science methodology is defined as the design and investigation of artifacts in a given context [36]. In particular, it is an iterative process in which researchers engage in several cycles of three main activities: problem identification and opportunities representation, development of solutions (i.e. software prototyping), and evaluation of the proposed solutions in a given context to figure out whether the solutions effectively accommodate the problem.

A conceptual research [37] was performed in Paper A (Chapter 2) where a vision on a new generation software design environment was presented. In particular, innovative ideas and new concepts were proposed to tackle the challenges (previously described in Section 1.1) that adversary affect the design process.

Gould and Lewis [38] defined three principles that were believed to lead to develop useful and easy to use computer systems: (i) early focus on users and tasks, (ii) empirical measurement, and (iii) iterative design by performing user tests, identifying and fixing problems, and observing effects of the fixes. For the development and evaluation of the proposed conceptual solution, *OctoUML*, the interaction design development approach was followed. As described by [39], the concern of interaction design is to design for a user experience by developing interactive systems that are usable. Interaction design consists of four main activities (See Figure 1.6): (a) identifying needs and establishing requirements, (b) developing alternative designs, (c) building a prototype of the system, and (d) evaluating the developed prototype. The purpose of using the interaction design approach is to make technology fit easily into peoples' activities, rather than forcing their activities to fit the dictates of technology [12].

In order to perform an early assessment of the different prototypes of *OctoUML*, we carried out an interdisciplinary collaboration with the Human-Computer Interaction (HCI) division at the University of Gothenburg. In particular, expert researchers in HCI gave their feedback and suggestions on how to improve the user experience, as well as the usability of the design environment.

User studies offer a scientifically sound method to evaluate the strengths and weaknesses of different visualization and interaction techniques, as well as to investigate social and cognitive processes surrounding them [40]. User studies were conducted in papers B, C, and D to evaluate the concept (i.e. *OctoUML*), as well as the functionalities and usability thereof. In brief, these studies were based on collecting both quantitative and qualitative data based on questionnaires that assessed perceived usability, efficiency and effectiveness. Often semi-structured interviews were also performed after the completion of the questionnaires.

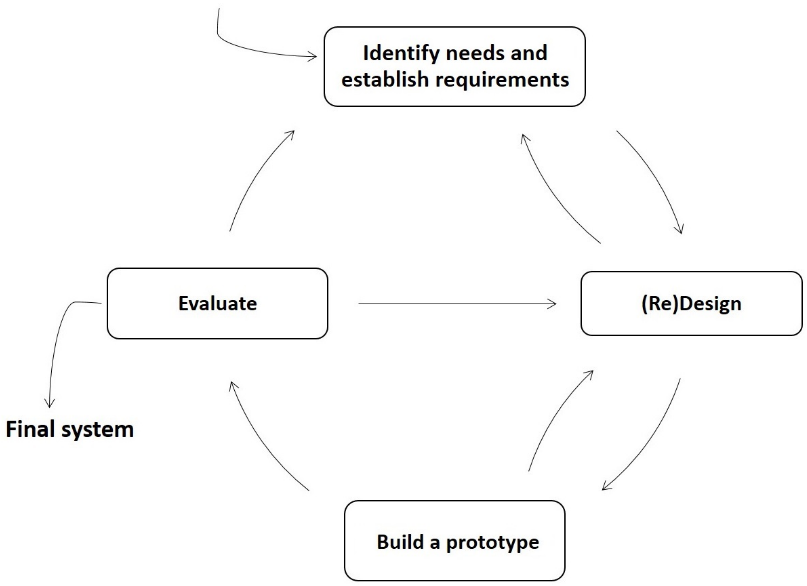


Figure 1.6: A simple model of the interaction design life-cycle

1.5.2 Empirical Methods

The aim of empirical research methods is to gain knowledge and increase understanding by observing and evaluating processes, software tools and human-based activities [41].

In paper C (Chapter 4), a comparative empirical study was conducted to compare two versions of *OctoUML*; one version is enabled to support voice commands and the other one is not. This was done to understand whether the efficiency of the software design process, as well as the tool usability, can be enhanced when supporting new interaction modalities.

Controlled experiments help to investigate a testable hypotheses where one or more independent variables are manipulated to measure their effect on one or more dependent variables [42]. In paper E (Chapter 6), a controlled experiment was conducted to dissect the design effort and drawing effort in UML modeling. The experiment was controlled in order to limit variables other than the chosen independent variables form affecting the experiment results. Furthermore, a subject questionnaire was used after the experiment in order to collect qualitative data regarding the experiences of the people who participated in the experiment.

1.6 Contributions

In this section the main contributions of the five papers included in this licentiate thesis (see Chapters 2, 3, 4, 5 and 6) are summarized.

1.6.1 Paper A: A Vision on a New Generation of Software Design Environments

This paper is a vision paper. The main contributions are based on answering the following research questions:

- **RQ.A1.** How can an integrated design environment having the power of both formal and informal tools be achieved?
- **RQ.A2.** How can modeling tools be easier to use and more productive?

In order to answer these research questions, some key aspects to generalize existing modeling tools are presented. In particular, the paper proposes that software design environments should:

- integrate informal and formal modeling notations,
- support multiple modes of interaction,
- support collaborative software design,
- integrate other software engineering tools, and
- be usable and deployable on multiple platforms.

In the next paragraphs, more details regarding each of these aspects are presented. In order to get further details, see Chapter 2 for the complete paper.

I. Mixing informal with formal notations. While exploring software design problems and discussing solution alternatives, software designers often go to the whiteboard and use informal notations in order to prototype their thoughts and ideas. This is because the whiteboard does not limit designers' expressiveness. CASE tools are usually used to formalize and document the design solution that are created on the whiteboard. There is a gap between early-phase software design exploration process where designers often use an informal tool (e.g. a whiteboard) and formalization/documentation process where a formal tool is used (e.g. a CASE tool).

As a remedy of the aforementioned gap, software design environments should support the creation of both informal and formal modeling notations. Furthermore, they should allow the transformation of informal models into formal contents using a recognition mechanism. This to:

- *let such contents be easily exchanged with other software tools, and*
- *enable the gradual transition into more detailed models used in later stages of development.*

II. Support for multiple modes of interaction. The interaction with the majority of CASE tools is based on using the mouse and keyboard. This actually limits the amount of interaction with the design environment when there is more than one designer involved in the software design activity. The interaction with whiteboards is intuitive. It is based on using the hands (holding a stylus or a piece of a chalk) for sketching. Moreover, during software design sessions, software designers usually use gestures by waving their hands or pointing to one part of the diagram in order to explain or communicate their ideas.

Some forms of disability prevent people from using their hands. In this case, interaction with software design environments via voice commands would re-mediate such inconvenience. Touch-based devices such as large touch screens and interactive whiteboards are intuitive and becoming common in class room environments, as well as in meeting rooms.

To support practical design sessions, software design environments should support multiple modes of interaction e.g. mouse, keyboard, voice, hand gesture and touch-input.

III. Supporting collaborative software design sessions. Software design is often a collaborative activity. Multiple designers meet *on-site* or remotely in order to discuss design problems and create design solutions. Whiteboards support collocated software design sessions, but do not provide means for supporting remote software design sessions between geographically distributed designers. CASE tools cannot be used by more than one designer simultaneously.

Software design tools should support both collocated and remote collaborative software design session.

IV. Integration of other software engineering tools. Software design affects and is also affected by other software engineering activities. A change in the requirements could lead to a changes in the design. Similarly, going with one design solution rather than with another one could affect the performance of the developed software system. Therefore, Software design environments should be integrated with other software management tools in order to provide effective support for software modeling across a project's lifecycle.

In particular, software design environments should be capable of addressing issues like requirements analysis and management, programming and coding, performance and security analysis, testing and model version management.

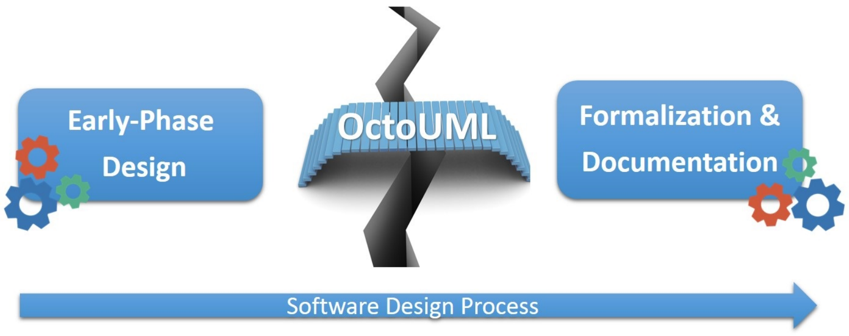


Figure 1.7: The main goal of *OctoUML*

V. Usability and multi-platform. The usability of current CASE tools is a common source of criticism. Indeed, these tools are difficult to use and not designed for a user experience. Moreover, the majority of CASE tools are designed to run on standard PCs and tailored to be used by one designer at a time.

Software design environments should rather be designed for an intuitive software modeling experience and deployable on multiple devices e.g. PCs, large touch-screens, tablets and interactive whiteboards.

1.6.2 Paper B: Towards a New Generation of Software Design Environments: Supporting the Use of Informal and Formal Notations with OctoUML

The contributions of this paper are two-fold: design and creation of a new generation software design environment, *OctoUML*, and a qualitative evaluation and accompanying usability and efficiency discussion of the environment.

To realize the vision that was described in Paper A, a new generation software design environment, called *OctoUML*, is created. *OctoUML* combines the advantages of both whiteboards and CASE tools, and supports the achievement of informal and formal modeling purposes (see Table 1.3).

Figure 1.7 shows the main goal of *OctoUML*. Indeed, the goal is to bridge the gap between early-phase software design process (when designers often do reasoning and sketch their ideas) and the formalization and documentation process (when CASE tools are usually used to document the designs).

OctoUML can be deployed on different platforms, e.g., desktop computers, personal computers, large electronic touch screens and interactive whiteboards. *OctoUML* enables users to create and mix both informal and formal modeling

Table 1.3: Informal and formal modeling purposes as supported by the functionalities of OctoUML

	Modeling Purpose					
Functionality	Ideation	Exploration	Collaboration	Communication	Documentation	Code Generation
Informal Notations (IF)	✓	✓	✓	✓		
Formal Notations (F)			✓	✓	✓	✓
Transition from IF to F					✓	✓
Integration of Analysis Tools		✓				
Version Management			✓		✓	
Multi-User Interaction	✓	✓	✓	✓		
Remote Interaction	✓	✓	✓	✓		
Saving Design as XMI					✓	✓
Sharing Design			✓	✓		

notations at the same time. It provides a selective recognition mechanism that is used to transform informal elements into formalized contents. Furthermore, *OctoUML* supports collocated software design sessions thanks to the adopted multi-touch mechanism. This mechanism actually enables more than one user to interact with the environment simultaneously.

A user study [40] was conducted in order to assess and evaluate the system. In particular sixteen subjects were engaged in a design assignment. After that, the subjects were asked to assess the usability using the SUS questionnaire [43]. Furthermore, the subjects were involved in semi-structured interviews in order to collect their feedback and perceptions. The following research questions were addressed:

- **RQ.B1** Does the proposed conceptual solution, *OctoUML*, provide a usable environment?

- **RQ.B2** Does support for mixing informal and formal notation better support the software design process?

The results show that *OctoUML* provides a usable and user-friendly environment. Moreover, *OctoUML* is perceived to have the potential to effectively support the activities of software designers by supporting the creation and mixing of both informal and formal modeling notations.

Chapter 3 provides more details on the characteristics and functionalities of *OctoUML*, as well as on the evaluation process and the obtained results.

1.6.3 Paper C: Interaction With Software Design Environments Via Voice For UML Design Support on Interactive Whiteboards: Insights And Experiences

Multi-modal systems use multiple integrated interaction modalities (e.g. sketch, touch, voice, etc.). Cohen and Oviatt [13] illustrated the importance of multimodal systems in reshaping daily computing tasks and predicted their future role in shifting the balance of human-computer interaction much closer to the human. To improve usability, efficiency and accessibility, we proposed in Paper A that software design environments should be equipped with microphones to record the spoken discussions, and have a recognition component to interpret users' voice-commands. To achieve such improvements, a voice recognition component was added into *OctoUML*.

The contributions of this paper are three-fold: the addition of a voice recognition system to *OctoUML*; the results of a qualitative evaluation of the voice recognition system and its functionalities; and the results of a quantitative comparative study in which the efficiency of two versions of *OctoUML* were compared. One version is equipped with the voice recognition system, *OctoUML-V* and the other one is not, *OctoUML-Lite*. The following research question was addressed:

- **RQ.C** Does the employment of voice interaction modality within the software design environments enhance the efficiency of the software design process?

The results of the evaluation show that:

- Generally all the functionalities of *OctoUML* were perceived suitable to be evoked by voice-commands. In particular, the task of using the keyboard for text input is perceived as most important to be supported by voice-commands. The main reason is that using a keyboard is not ergonomic and a time-consuming task.
- The perceptions regarding the usability of the voice interaction modality supported by *OctoUML* are positive. Moreover, *OctoUML-V* is perceived more enjoyable than *OctoUML-Lite*.
- The employment of voice interaction modality within the software design environments enhances the efficiency of the software design process by reducing the time required for text input.

Chapter 4 provides more details on the motivation for adding the voice recognition system to *OctoUML*, the types of voice-commands that are supported, and the evaluation process together with the obtained results.

1.6.4 Paper D: OctoUML: An Environment for Exploratory and Collaborative Software Design

This paper is a demonstration paper of *OctoUML*. The contribution of this paper is the demonstration video which was specially recorded to disseminate the concept of *OctoUML*. A demonstration is an effective way to show the idea and functionalities of a system in action in order to enable people to fully grasp its value and potential. In this paper the architecture and functionalities of *OctoUML* are presented, and the design process is described with a scenario. Furthermore, the results of the evaluations performed in Papers B and C are re-presented in this paper. The demonstration video can be viewed via the following link: `https://youtu.be/fsN3rfEAYHw`

1.6.5 Paper E: Dissecting Design Effort and Drawing Effort in UML Modeling

Model-based approaches are considered to be beneficial in improving software development productivity and product quality (e.g. [15]). However, these approaches are criticized of the supposedly large effort they require to do modeling. Regardless of its reported benefits, software modeling is still believed as an unnecessary approach by some developers [17].

In this paper, the process of software modeling is analyzed in order to find out: (i) how much effort is spent on the design of a solution (i.e., thinking and making design decisions), and (ii) how much effort is spent on drawing of the design solution with a modeling tool (i.e. tool interaction). In particular, the modeling activity is considered to consist of three main cognitive sub-activities:

- *Design*: ideation and thinking about the design.
- *Notation Expression*: representation of a design via the modeling notation.
- *Layout*: spatial organization of the elements of a model.

The drawing effort is the effort spent on *notation expression* and *layout*. The following questions were addressed:

- **RQ.E.** How can the design effort and drawing effort in software modeling be dissected?

  RQ6.E1. How much of the modeling effort is spent on design?

  RQ6.E2. How much of the modeling effort is spent on drawing the design solution?

To compute the effort spent in each sub-activity, two-phase experiments were conducted. In the first phase, the effort required to make the initial model of a system (modeling effort) was measured. In the second phase, the effort required to recreate the same model again is measured, simply by redrawing the already defined solution (copying effort). At the end, the design, notation expression, layout efforts are calculated by assessing the time difference between the two phases.

The initial findings suggest that the majority of the modeling effort is devoted to the design (design effort). This means that projects that create models incur at least significant thinking about the design. However, the effort spent on using modeling tools (drawing effort) could be reduced by investigating better modeling-tool support.

Chapter 6 provides more details on the design of the controlled experiments, as well as on the approach which is used to calculate the effort of each modeling sub-activity.

1.7 Conclusion

The overall goal of the PhD research is to better support a realistic software design process by understanding software design practices, as well as proposing and assessing a new generation of software design environments. As a first steps towards achieving the overall goal of the research, the following four goals are addressed in this licentiate thesis:

First, a number of challenges that adversely affect the process of software design were discussed. Such challenges are mainly related to current modeling tools, as they lack of proper support of the software design process. To overcome the challenges and effectively support the software design process, a vision on a new generation software design environments was presented in Paper A (*Goal 2*). In particular, software design environments are proposed to integrate informal and formal modeling notations, support multiple modes of interaction, support software design collaboration sessions, integrate other software engineering tools, and be usable and deployable on multiple platforms.

Second, a new generation environment for exploratory and collaborative software design, called *OctoUML*, was designed and subsequently created (*Goal 3*). Table 1.4 provides a comparison between the characteristics and supported functionalities of informal tools, formal tools and *OctoUML*. *OctoUML* is an open source system. It supports multiple modes of interaction such as mouse, keyboard, touch and voice-commands. It also supports the creation of both informal and formal modeling notations simultaneously. To open up new opportunities for interactive collaborative design, *OctoUML* is enabled to support both collocated and remote collaborative design sessions.

Table 1.4: Comparison between the characteristics and supported functionalities of informal tools, formal tools and OctoUML

Functionality	Informal Tools	Formal CASE Tools	OctoUML
Informal Notations (IF)	✓		✓
Formal Notations (F)	✓	✓	✓
Transition from IF to F			✓
Version Management		Sometimes (ST)	✓
Integration of Analysis Tools			✓
Collocated Design	✓		✓(multi-touch)
Remote Design		ST	✓
Saving Design as XMI		✓	✓
Design Sharing Electronically		✓	✓
Characteristic			
Simplicity	✓		✓
Flexibility	✓		✓
Designed for UX	✓		✓
Ease of Use	✓		✓
Ease of Learning	✓	ST	✓

Third, the efficiency and usability of *OctoUML* was assessed through user studies (*Goal 4*). The perceptions regarding the usability and efficiency of *OctoUML* were positive. Furthermore, *OctoUML* was perceived to have the potential to effectively support the activities of the designers.

Fourth, two experiments were conducted to increase the understanding of the software design process (*Goal 1*). In particular, the software modeling process was analyzed in order to dissect the effort spent on thinking about the design (*design* effort) and the effort spent on drawing the design solution in a modeling tool (*drawing* effort). It turned out that the majority of the modeling effort is dedicated to design thinking, whereas the effort dedicated to draw the design solution, i.e. tool interaction, cannot be neglected. To reduce such an effort, better modeling-tool support should be investigated.

1.8 Future Work

The goals of this licentiate thesis are considered as first steps towards achieving the overall goal of the PhD research (see Section 1.2). In this section, the future directions are presented and discussed (see Figure 1.8).

The first direction is to enrich the body of knowledge regarding the software design practices. With regards to the estimation of *design* and *drawing* efforts, more experiments will be conducted to increase the validity of the results already obtained so far. Some aspects that will be investigated are: software design experience, professionalism, modeling tools and modeling languages.

The second direction is to perform a further assessment of the functionalities of *OctoUML*, as well as to investigate new approaches for supporting the software design process. More details are presented in the next two paragraphs.

Further assessment of OctoUML. Software design is usually a collaborative activity. In fact, more often than not several designers work together to understand a domain problem and come up with a design solution. *OctoUML* is enriched with a multi-touch mechanism by which more than one designer can interact with interface simultaneously. Moreover, *OctoUML* allows multiple designers geographically distributed to collaborate remotely. To investigate collaborative design and modeling in software engineering, several collaborative design sessions will be conducted and observed. Substantially, the goals are to: (i) acquire knowledge to build better software design tools, (ii) understand the effect of modeling in distributed design, and (iii) explore and understand the choice of formality in distributed design.

Navigation and Understanding. Models are characterized of being high-level abstraction software artefacts. In contrast, code fragments are characterized of being low-level abstraction software artefacts. In MDE, models are mainly used to generate code fragments. Most of the tools that enable forward, reverse and round-trip engineering lack support for an effective visualization of, navigation and linking between models and code. As a remedy, novel approaches to visualize models and code via *OctoUML* will be investigated and provided. The main goals are to: (i) reduce the gap in abstraction level between models and code, (ii) support traceability and navigation between the two artefacts, and (iii) support overall software comprehensibility.

Analysis. Another direction is to integrate other software engineering tools within *OctoUML*. For example, Smith and Williams [45] state that performance is a critical quality for the success of today's software system. A performance analysis tool will be integrated with *OctoUML*. The overall goal is to give a view on the performance of software systems during early-phase design processes, when design choices are usually explored, confronted and made.

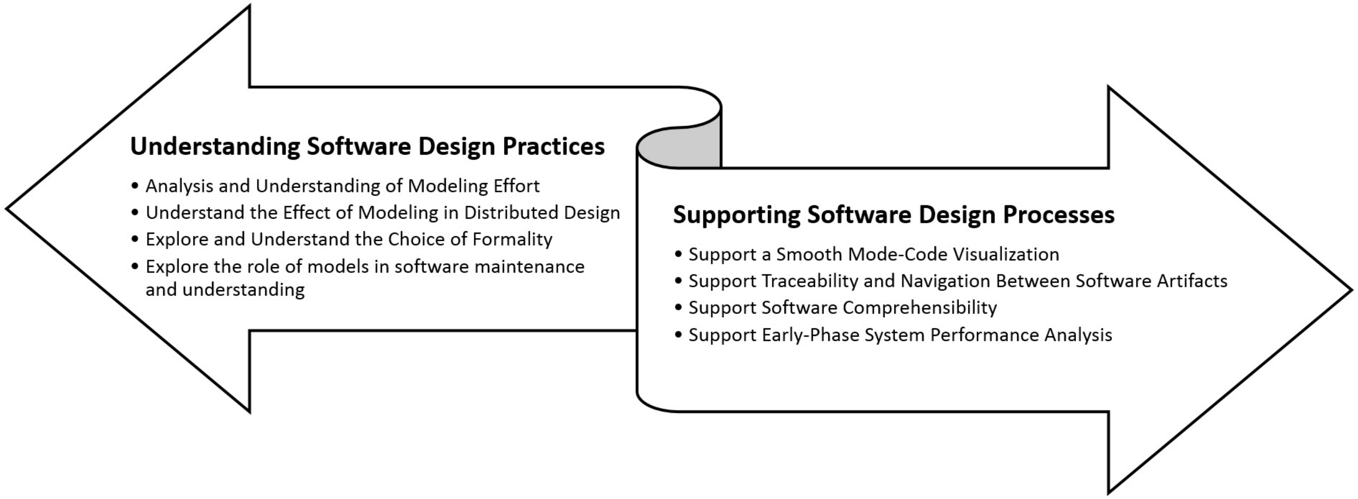

Understanding Software Design Practices
• Analysis and Understanding of Modeling Effort
• Understand the Effect of Modeling in Distributed Design
• Explore and Understand the Choice of Formality
• Explore the role of models in software maintenance and understanding
Supporting Software Design Processes
• Support a Smooth Mode-Code Visualization
• Support Traceability and Navigation Between Software Artifacts
• Support Software Comprehensibility
• Support Early-Phase System Performance Analysis

Figure 1.8: Future Directions

Chapter 2

Paper A

A Vision on a New Generation of Software Design Environments

M.R.V. Chaudron, R. Jolak

In First International Workshop on Human Factors in Modeling (HuFaMo 2015). CEUR-WS, pp. 11-16. 2015.

Abstract

In this paper we illustrate our vision for a new generation software design environment. This environment aims to generalize existing modeling tools in several ways - some key extensions are: integration of rigorous and informal notations, and support for multiple modes of interaction. We describe how we can consolidate the environment by integrating it with other software engineering tools. Furthermore, we describe some methods which could permit the environment to provide a flexible collaborative medium and have a practical and inspiring user experience.

Keywords: Software Engineering; Modeling Tools; Collaborative Design; IDE.

2.1 Introduction

Software systems have an important role in the technological evolution which we are witnessing nowadays, and as a consequence, software systems are becoming more and more complex. The increasing complexity of such systems has raised some certain challenges, such as e.g. design uncertainty and run-time changes, making it difficult to meet continuous customer demands for a better software quality [46, 47]. Software modeling plays a pivotal role in software development. Models present an understandable description of complex systems at several levels of abstraction and from a diversity of perspectives. Furthermore, they provide an essential medium matching between problem and software implementation by describing users needs and prescribing the product to be developed.

Software modeling is a highly complex and demanding activity [32]. Software designers often use software modeling tools to perform a software design. There are two dimensions of these tools that we will challenge in this paper: i) the formality of the notation used, and ii) the modes of interaction supported by the tools. Next, we briefly explain our views on these dimensions.

First, we classify modeling tools into two groups: informal and formal. We mean by informal any tool that supports informal design in the sense that it does not constrain the notation used. Examples of such tools are whiteboards, paper and pencil. Whiteboards are often used to collaboratively sketch software modeling ideas, discover architectural solutions, capture design discussions, etc. [2, 48]. Whiteboards are normally used for sketching when more than two people are involved [10]. Generic diagramming tools such as PowerPoint and Visio are informal in the sense that they do not constrain the notation, but they do provide mature digital editing functionality (move, delete, undo). While on the other hand we mean by formal tools any CASE tool which supports one or more formalized notations. Typical examples are UML CASE tools (like Rational Rose, Enterprise Architect, Visual Paradigm, StarUML etc.). Also for many other modeling languages, tools are often dedicated to a single notation (Archimate for Enterprise Modeling, ARIS-tool for Business Process Modeling, etc.). All CASE tools support mature digital edition functionality.

Table 2.1 is based on Hammouda [9] and describes some relative advantages of informal and formal modeling tools. We envision our environment to have the advantages of both formal and informal tools.

The second dimension we challenge is that of the modes of interaction supported by modeling tools. Oviatt and Cohen [13] illustrated the importance of multimodal systems in reshaping daily computing tasks and predicted their future role in shifting the balance of human-computer interaction much closer to the human. Based on that, we want to support multimodal communication interactions by recognizing touch, voice and gesture for a more intuitive software modeling experience.

Table 2.1: Relative Advantages of Informal and Formal Modeling Approaches

	Informal	**Formal**
Clarity		High clarity because of strict adherence to syntax
Flexibility	Caters for improvisation of notation	
Ease of continuous design	In tools based on digital editing, editing (move, resize, delete, undo, etc.) is easier than in sketch-based tools such as whiteboards.	
Ease of learning Notation		Formal syntax checking helps in learning the proper syntax
Intuitiveness of using tool	Very simple to use; but limited in functionality	More difficult to learn, but advanced functionalities supported
Collaboration	Multiple people collaborating on a shared design prefer to use informal representations [10]	
Integration	Absence of a formal syntax (and semantics) prohibits exchange of designs	Formal syntax allows a formal representation of the design that can be exchanged with other tools

Summarizing, the following questions are addressed:

- **Q.1.** How can we achieve an integrated design environment having the power of both formal and informal tools?
- **Q.2.** How can we make modeling tools easier to use and more productive?
  - How can tools better support tasks of software developers? Our focus is on tasks related to the design of systems.
  - Which sources of knowledge and information can be connected to provide information needed at easy disposal (right information at the right moment, place and format)?

The paper is organized as follows: in Section 2.2 we describe the related work. Section 2.3 illustrates our vision. Finally we conclude and discuss ideas for future work in Section 2.4.

2.2 Related Work

Many empirical studies of formal tools usage have pointed out that software designers consider these tools overly restrictive and this often lead to poor utilization [48, 49]. By doing a HCI study, Plimmer et al. [50] revealed that in early software design phases, the designers prefer to sketch by hand rather than using a keyboard or a mouse. Whiteboards support informal design. They are frequently used by software designers during project meetings to sketch ideas and thoughts about system goals, requirements and design solutions [2, 48].

Electronic interactive whiteboards offer the potential for enhanced support by allowing the manipulation of the content, handling of sketches, and doing collaborative distributed works. Mangano et al. [51] identified some behaviors that occur during informal design. They implemented an interactive whiteboard system to support these behaviors, and identified some ways where interactive whiteboards can enable designers to work more effectively. The main goal of the system that they implemented, called Calico, is to maintain fluidity and flexibility allowing software designers to focus on the content of their sketches rather than the tool used to make it. Mangano et al. [51] revealed a number of weaknesses in Calico ranging from usability issues to challenges inherent to interactive whiteboards. In particular, designers reported that some gestures were not rapidly interpreted, and the large e-whiteboard diminished the quality of their handwriting, forcing them to write slower or larger. We want to support software design not only with interactive whiteboards, but with PCs, touch pads and smart phones.

Baltes and Diehl [10] investigated the use of sketches in software engineering activities by conducting an exploratory study in three different software companies. Their results showed that the majority of the sketches were informal, and the purposes of sketches were related to designing, explaining, or understanding. Baltes and Diehl also showed that the sketches were archived digitally for re-visualization and future use. Like us, they think software design tools should enable informal design sketching.

West et al. [52] stated that software engineers often use paper and pencil to sketch ideas when gathering requirements from stakeholders, but such sketches on paper often need to be modelled again for a further processing. A tool, FlexiSketch, was prototyped by them to combine freeform sketching with the ability to interactively annotate the sketches for an incremental transformation into semi-formal models. The users of FlexiSketch were able to draw UML-like diagrams and introduced their own notation. They were also able to assign types to drawn symbols. Users liked the informality provided by the tool, and stated that they would be willing to adopt it in practice. FlexiSketch runs on tablet computers. It is a single user tool, and does not support collaborative sketching. We think running FlexiSketch on electronic whiteboards could allow for multi-user input and facilitate collaboration. We also think that software design tools should be able to support sketch recognition and its transformation into a kind of formal diagrams as well as allow the exportation of such diagrams to other programs e.g. CASE tools. FlexiSketch Team [53] is an extended version of FlexiSketch, which supports a collaborative sketching via ad-hoc local Wi-Fi network, but it does not allow for a distributed collaboration.

Chen et al. [54] have developed SUMLOW. A sketching-based UML design tool for electronic whiteboard technology. It allows to preserve hand drawn diagrams and supports for manipulation of the diagrams using pen-based actions. UML sketches can be formalized and exported to a 3rd party CASE tool. Their tool does not support design sketching on different platforms like mobiles and tablets. Again as all works previously mentioned, it does not support a collaborative distributed software modeling.

MaramaSketch [55] includes a meta-modeling editor. This editor allows to define a conventional modeling language which is then used to compile a modeling tool for the defined language. However, MaramaSketch needs to create the complete language definition first, and after that, users must strictly follow it. So as a consequence, it prevents any flexible sketching.

Magin and Kopf [56] have created a multi-touch based system allowing users to collaboratively design UML class diagram on touch screens. They have also implemented new algorithms to recognize the gestures drawn by the users and to improve the layout of the diagrams. However, it does not support a remote collaboration, and as they stated, their tool has some usability challenges in creating and editing of sketches, and in the recognition process of hand written text.

In the area of integration of software development tools Open Services for Life-cycle Collaboration OSLC [57] is an emerging standard. This standard defines API's through which development tools can interoperate. OSLC could be a technology that underlies the integration aspects of our vision.

Brosch et al. [58] showed the importance of model versioning in enabling efficient team-based development of models. Based on that, we think a version management tool should be integrated within software design environments to track modeling processes and their evolution. There is a fair amount of work ongoing in versioning for software models [59], but none of this has been integrated in mainstream CASE tools yet.

IBM integrated development environments [60] allow for a collaborative software development. In particular, they provide teams with rich capabilities for continuous developing, testing, analyzing and optimizing applications. For example, IBM Rational Business Developer is an Eclipse-based environment. It allows complex applications to be modeled graphically and simplifies their development. Anyway, IBM does not permit for a free access to these environments.

While a comprehensive theory of IDE's does not exist, there are proposals for theoretical models that can serve to support the design of NGDE. Notable approaches are the Cognitive Dimensions approach [61] and the 'Physics of Notations' [62].

2.3 Our Vision

In this section we present our vision for a more intuitive, inspiring and efficient tool to support exploratory and collaborative software modeling. In particular, we are going to describe some ideas which we consider to be relevant to achieve such a tool. We will refer the next generation software design environment as NGDE.

One first point is that in practice modeling and designing go hand-in hand. A modeling language provides the notation in which to express a design. As such a modeling language is a part of the toolbox that a designer uses in the creative process of designing a solution. Currently, most case tools are modeling (or even diagramming) tools. Instead, the next generation of tools should take a holistic view on supporting all design activities in which developers are engaged.

Next, we discuss several key aspects in which NGDE can provide better support for the design activities of software developers.

2.3.1 Informal Versus Formal Notation

Informal tools like whiteboards provide a useful mean for flexible collaborative modeling. In fact, software designers can easily create/extend diagrams, add comments and highlight some parts of their sketches. Even more, they can sketch diagrams of multiple notations without following any restrictive rule imposed by the formality of a one modeling language or syntax. However, re-modeling is a difficult and a time consuming task. Moreover, whiteboards do not support data persistence and transference. Formal tools like CASE tools are restrictive in that they require designers to use some specific notations for modeling. We propose that NGDE tools should support the mix of both formal and informal modeling notations that designers use. Ideally, NGDEs should maintain the characteristics of formal tools in their support of design transfer and persistence. To support informal modeling, tools should allow designers to create different types of diagrams on the same canvas [51]. Furthermore, they should not constrain designers to sketch only some specific notations. For instance, designers should be able to draw and create a variety of sketches e.g. use case diagrams, work-flows, arrows, state charts, data models, etc. In general, NGDE should enable designers to add domain specific icons or notations. These kind of notations help to better describe a specific domain problem.

NGDEs should keep from existing editors the abilities to organize diagrams by moving, resizing, grouping and separating sketches, and the ability to modify and evolve sketches.

NGDEs should have the ability to transform sketches into formalized content by providing a recognition unit. This enables designs be formally represented and hence easily exchanged with other software tools. Furthermore, they should have the power of formal tools in maintaining and transferring the designs for further processing tasks.

2.3.2 Integration

In their daily work, many software designers work concurrently on different artefacts: changes to a design and followed closely by changes in code and changes in requirements. Unfortunately, with current tools the developer needs to switch to different applications. We propose to design an integrated environment. This does not need to become a 'Swiss army knife' that integrates all functionality in a single tool. Integration of development tools needs to address a shared data model of software development artefacts, but also a shared UI-concept.

The goal behind the idea of integrating other software management tools within a software modeling tool in a 'one stop' environment is to provide effective support for an effective software modeling process.

Software requirements, for instance, evolve over time and they are frequently subject to changes during initial development and later on to delivery. Designers generate many ideas in order to understand a problem and find a solution for it. These ideas are often compared, modified, evaluated and enriched as the modeling process evolves. In order to realize how useful could be having a trace of the requirements within a software modeling tool, let us think about the following scenario: a group of software designers start to document and gather needs of a specific software product, after that, they proceed to create a first design of the product using for example a traditional whiteboard or a CASE tool. Let us also consider that the software needs, written on a simple paper sheet, are given by a client. In both cases, using either a whiteboard or a CASE tool, software designers have to meet again and again whenever new requirements come out or having earlier requirements exposed to changes. This is a time consuming task and especially when designers have to recollect their designs and re-model them according to the new set of requirements.

Here, a NGDE should be able to handle notes written on paper as an artefact. Ideally, this paper is not only stored as a (jpeg) picture, but contents are (partially) recognized and can be transformed into formal concepts.

Modeling involves several stakeholders who conduct the creation of the design in elicitation and formalization phases, and since requirements evolve over time, modeling usually comprises several iterations of elicitation and formalization resulting in an evolving process [63]. Therefore, we think that software modeling tools should be integrated with other software engineering tools to deliver a 'one stop' environment capable of addressing and supporting issues like requirements analysis and management, programming and coding, generation of bug reports, performance and security analysis, testing, versioning, etc. This is in line with the Twin Peaks model by Nuseibeh [64]. This model states that in reality requirements and designs develop progressively in concert and mutually influence each other.

Of course, source code is also essential in the development of software. Hence throughout the development process, models and code must be combined - in the sense that developers must be able to view them side by side and jump between editing one and keeping the other synchronized. One challenge is the linking between models and code. It is typical that models are used at various and varying levels of abstraction. Models start out at a high level of abstraction and gradually get refined by adding details.

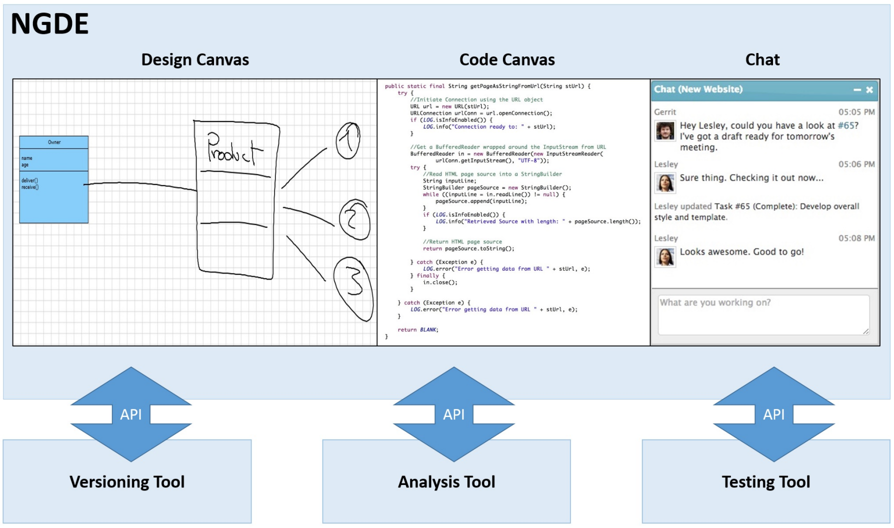
The integration of code and models also raises the question of debugging. While we can always use the IDE's debugging functionality, it usually does not make sense to debug the generated code itself. The developer is more familiar with the model than the code, and if a problem is found, we want to correct it in the model not the code. Modeling tools should support the usage of models in debugging the functionalities at a higher level of abstraction in order to know if the application is doing what it was designed to do.

Modeling tools should support multiple people and teams working on the same design from different locations. In particular, they should provide means to achieve an effective coordination between geographically distributed project members. Version management, for example, should be adapted to support collaborative modeling and design. We think modeling tools should provide a repository to keep track of the version history of the models stored in it, as well as provide the ability to observe who is changing what artifacts in the environment. Following the checkout-update-commit interaction paradigm, the repository would offer an interactive model merging tool to resolve conflicts when two users change the same model data. It would increase the potential for parallel and distributed work, improve ability to track and merge changes over time and automate management of revision history. It would also allow multiple designers to work with the same models concurrently, supporting tight collaboration and a fast feedback loop.

Finally, and to provide an effective communication medium for a geographically distributed software modeling teams, we propose integrating social media and chat tools within the modeling tools. The goal is to make software modeling activity more efficient. For instance when two designers from two different locations want to exchange ideas about a specific design, they could make use of the integrated social media or chatting tools to do such a task, and of course, this reduces the time spent handling emails. In fact, these communicative facilities play an important role in establishing a basis for discussions and negotiations, information exchanging, and sharing data e.g. images, videos, etc.

In general, modeling tools should be "open" providing various integration mechanisms among the different platforms. In particular, they should have programmable interfaces, import/export formats, and enable plug-ins for integration (see Figure 2.1), thus offering an ideal support for team work, and letting the overall development process becomes easier and faster. On this topic, we will explore to what degree OSLC helps solve this issue. In summary, integration has several facets:

- Integration of rigorous and informal notation,
- Integration of different tools for different activities in the software development life-cycle,
- Integration of (machine and human) knowledge sources.



```
public static final String getPageAsStringFromUrl(String stUrl) {
    try {
        //Initiate Connection using the URL object
        URL url = new URL(stUrl);
        URLConnection urlConn = url.openConnection();
        if (LOG.isInfoEnabled()) {
            LOG.info("Connection ready to: " + stUrl);
        }

        //Get a BufferedReader wrapped around the InputStream from URL
        BufferedReader in = new BufferedReader(new InputStreamReader(
                urlConn.getInputStream(), "UTF-8"));
        try {
            //Read HTML page source into a StringBuilder
            String inputLine;
            StringBuilder pageSource = new StringBuilder();
            while ((inputLine = in.readLine()) != null) {
                pageSource.append(inputLine);
            }
            if (LOG.isInfoEnabled()) {
                LOG.info("Retrieved Source with length: " + pageSource.length());
            }

            //Return HTML page source
            return pageSource.toString();

        } catch (Exception e) {
            LOG.error("Error getting data from URL " + stUrl, e);
        } finally {
            in.close();
        }

    } catch (Exception e) {
        LOG.error("Error getting data from URL " + stUrl, e);
    }

    return BLANK;
}
```

Chat (New Website)

Gerrit 05:05 PM
Hey Lesley, could you have a look at #65? I've got a draft ready for tomorrow's meeting.

Lesley 05:06 PM
Sure thing. Checking it out now...

Lesley updated Task #65 (Complete): Develop overall style and template.

Lesley 05:08 PM
Looks awesome. Good to go!

What are you working on?

Figure 2.1: An illustrative view of NGDEs and the integration mechanism

2.3.3 Usability, Interaction and Collaboration

The usability of current CASE tools is a common source of criticism. One key aspect of usability is the manner in which humans interact with the system. Currently this is by using the keyboard and the mouse - essentially we are using the computer as an enriched typewriter. Other modes of interaction have gained popularity because of their intuitive nature and these should also be employed in the area of software design environments.

Touch-based interfaces have become common in tablets and smart phones, and also smart-whiteboards with touch-based interaction have been introduced in class room environments. This introduces the dimension of modality of interaction. While traditional interaction with a computer is via a keyboard (and mouse) currently there are many options available: voice, touch, gesture, eye-focus or laser-focus as pointing. Computers are capable of handling such new types of inputs. This will make interacting with NGDEs much more intuitive.

Whiteboards allow multiple users to draw software models together. In order to emulate this informality in our environment, we propose enabling it to support a collaborative multi-touch modeling. Multi-touch is an interaction technique that permits the manipulation of graphical entities with several fingers at the same time. Making use of multi-touch screens allows users to design complex diagrams simultaneously by performing simple intuitive touch gestures to draw their part of the diagram.

Such joint drawing sessions typically also trigger a lot of discussion. Such discussion may contain valuable information about a design, such as e.g. its rationale. However, traditional tools do not capture the discussion. NGDE can be equipped with microphones and also record the spoken discussion. New challenges in this area will be to search through this type of recorded spoken text. Inspiring work in this direction is the work by Nakakoji et al. [65]. They describe a system that makes video-recordings of the design discussion in front of the whiteboard. Their system does automated voice recognition and produced a textual transcript of the discussion. Also, their system offers a way of navigating through the discussion using a time-line. The recording of discussions is effortless for (i.e. without any explicit action) the developers. In such a way NGDE can relieve the developer by lowering the cognitive attention needed for inputting relevant information into the system and linking it to related artefacts.

A task that is commonly forgotten is that of design review. Designers frequently review the design progress in order to know what is done and what they have still to do. For that, NGDEs should support design review "on the fly", as well as in detail whenever designers want to add some additional items to their previously sketched design. One approach in this direction is offered by the recent version of the Altova UModel tool. This tool provides a layering-mechanism: here review comments are part of one layer and the software design is part of another layer. The user can then select to see combined layers or layers in isolation.

2.3.4 Multi-platform

Rather than tying the development process down to any specific hardware environment, the NGDE should aim to facilitate multiple platforms: smart whiteboards, tables, smart phones, and traditional desktops and laptops. This increases the accessibility of the environment. Also, in classroom environments, this will open up new opportunities for interactive collaborative design.

2.4 Conclusion

Other application domains have gone before software development CASE tools and have shown that rich interaction with computer-based systems is enhancing productivity [13]. Next generation software design tools must keep up with this trend and offer improvements over existing tools in the following dimensions:

- Rich support for multiple modes of interaction (touch, audio, video, gesture),
- Support for mixing formal notations with informal notations (e.g. UML diagrams with additional sketching),
- Higher level of integration of tools: on the one hand integrating tools for different development tasks (requirements, testing, coding) and also analysis tools (performance, security). On the other hand, integration of social/organizational sources of knowledge via ((video) chat),
- Rich support for multiple platforms: work does not only happen behind a PC, there is also discussions at the whiteboard and via tablets. NGDE should offer a seamless environment for this.

The design of software design environments should be driven by studying the needs of actual software developers. We consider it very important that more observation studies are performed about the actual tasks that software developers perform. From this we can learn how to best support them.

This paper describes our vision. It is beyond our own capacity to realize this vision. We therefore call on the community to collaboratively work on the next generation of software design environments.

Chapter 3

Paper B

Towards a New Generation of Software Design Environments: Supporting the Use of Informal and Formal Notations with OctoUML

R. Jolak, B. Vesin, M. Isaksson, M.R.V. Chaudron

In Second International Workshop on Human Factors in Modeling (HuFaMo 2016). CEUR-WS, pp. 3-10. 2016.

Abstract

Software architects seek efficient support for planning and designing models. Many software design tools lack flexibility in combining informal and formal design. In this paper, we present OctoUML, a proof of a concept of a new generation software design environment that supports an effective software design process. The system provides options for collaborative software design and different input methods for creation of software models at various levels of formality. The design and architecture of OctoUML are also presented. The evaluation shows that OctoUML provides a user-friendly environment and has the potential to effectively support the activities of the designers.

Keywords: Sketching; Informal and Formal Desing; Recognition; Multi-touch; Design Environment, UML.

3.1 Introduction

As software systems are gaining increased complexity, the importance of efficient software design tools is also increasing. Software models change frequently and are quite often updated by many designers simultaneously [66]. These models should present a description of systems at multiple levels of abstraction and from different perspectives. Therefore, it is crucial to provide software design tooling that provides possibilities for efficient and collaborative development as well as options for creation and evolution of software models and architectures.

Sketches, or depictive expressions of ideas, pre-date written languages by thousands of years [67]. Many designers tend to sketch their initial ideas on the whiteboard [68]. These sketches present an intuitive way to prototype, communicate and record their thoughts. Sketches can facilitate discovery of new objects and foster new design ideas [69]. They effectively support the process of software design and serve designers to inspect and develop one design idea as well as reflect on some other alternatives [70].

On the one hand, whiteboards are commonly used for sketching initial software design, quite often by many people simultaneously [10]. They support informal design and do not constrain the notation being used. However, standard whiteboards lack of integration with subsequent software elaboration tools. Hence, re-modelling is difficult and a time-consuming task. On the other hand, CASE tools (e.g. StarUML, Rational Rose, Enterprise Architect, etc.) provide means to store and modify designs. However, they support one or more formal notations and hence restrictively require designers to use those specific notations for modelling. Actually, designers often sketch and use ad hoc notations that rarely adhere to standards like the Unified Modelling Language UML [18].

In previous work, we presented our vision on a new generation of software design environments [71]. One of the characteristics which we proposed for such environments is that they should be capable of supporting both informal and formal modelling. In other words, they would combine the advantages of both whiteboards and CASE tools, and therefore be able to bridge the gap between early design process (when designers often sketch their ideas) and formalization/documentation process. To realize our vision, we developed a software design environment called OctoUML[1] [72]. This environment can be run using a number of input devices ranging from desktop computers over large touch-screens to large electronic whiteboards. It allows simultaneous creation of both informal freehand sketches (using fingers or styluses) and formal computer-drawn notations, e.g., UML class diagrams. We have enriched our tool with some features, functionalities and services in order to support designers' activities and provide a practical and inspiring user experience.

[1]demo video: https://goo.gl/PmuUf8

Further details on these aspects will be described in Section 3. To assess our concept, we asked some subjects to evaluate OctoUML and give feedback on its usability. The following questions are addressed:

- **R.Q.1** Does our tool provide a usable environment considering issues like ease of use, efficiency and user satisfaction?
- **R.Q.2** Does support for mixing informal and formal notation better support the software design process?

This paper is organized as follows: Section 3.2 describes the related work. Illustration of our approach for supporting exploratory and collaborative software design is presented in Section 3.3. We provide the design architecture details of OctoUML in Section 3.4. Evaluation and results are presented in Section 3.5. We discuss the results in Section 3.6. Threats to validity are presented in Section 3.7. Finally, we conclude the paper and illustrate our plan for future work in Section 3.8.

3.2 Related Work

CASE tools support software development activities such as diagramming, code generating and documentation [49]. However, software designers consider such formal tools overly restrictive and limitative of their expressiveness [49, 54, 73]. Whiteboards are rather simple to use. In fact, they are frequently used by software designers due to their role in promoting creativity, idea generation and problem solving [2]. Electronic whiteboards provide the perspective for better software design support by permitting the management, control and maintenance of the contents [51, 54].

Mangano et al. [51] identified some behaviors that occur during informal design. In particular, designers sketch different kind of diagrams (e.g. box and arrow diagrams, UI mockups, generic plots, flowcharts, etc.) and use impromptu notations in their designs. The authors implemented an interactive whiteboard system (called *Calico*) to support these behaviors and identified some ways where interactive whiteboards can enable designers to work more effectively.

Wüest et al. [74] stated that software engineers often use paper and pencil to sketch ideas when gathering requirements from stakeholders, but such sketches on paper often need to be modelled again for further processing. A tool, *FlexiSketch*, was prototyped by them to combine freeform sketching with the ability to annotate the sketches interactively for an incremental transformation into semi-formal models. The users of *FlexiSketch* were able to draw UML-like diagrams and introduced their own notation. They were also able to assign types to drawn symbols. Users liked the informality provided by the tool, and stated that they would be willing to adopt it in practice.

Chen et al. have developed SUMLOW [54], a sketching-based UML design tool for electronic whiteboard technology. It allows the preservation of hand-drawn diagrams and supports the manipulation of them using pen-based actions. UML sketches can be formalized and exported to a 3rd party CASE tool.

Damm et al. [49] conducted user studies in order to understand the practice of software modelling. They observed that designers alternate between whiteboards and CASE tools, extend the semantics of the notations to support the design activities and allow expressiveness, sketch new ideas informally, and actively collaborate when they work in teams. The authors considered that a usable modelling tool should be designed to come across the aforementioned observed behaviours. They developed a tool called *Knight*. *Knight* supports informal and formal modelling using gestures on an electronic whiteboard. In order to achieve intuitive interaction, *Knight* uses composite gestures and eager recognition of hand-drawn elements. Damm et al. showed that informal drawings were temporary and usually erased after producing the formal diagram.

Magin and Kopf [56] created a multi-touch based system allowing users to collaboratively design UML class diagrams on touch-screens. They have also implemented a new algorithm to recognize the gestures drawn by the users and to improve the layout of the diagrams. However, their tool does not allow for informal freehand sketching of arbitrary notations.

Baltes and Diehl [10] examined the usage of sketches in software engineering activities by conducting an exploratory study in software companies. The results showed that the majority of the sketches were informal, and the purposes of sketches were related to modelling, explaining or understanding. Baltes and Diehl also revealed that the sketches were archived digitally for re-visualization and future use. Like us, they think software design tools should enable informal design sketching.

3.3 Approach

Our motivation for creating OctoUML is to provide a more intuitive, inspiring and efficient tool to support exploratory and collaborative software design. Key innovations of our approach are: (i) enabling users to create and mix both informal hand-drawn sketches and formal computer-drawn notations at the same time on the same canvas, (ii) providing a selective recognition mechanism that is used to transform hand-drawn sketches into formalized contents, and (iii) enabling of multi-user support on a single input device. In the next subsections, we describe those novel aspects in more detail. Table 3.1 shows the differences between our approach and the related work.

Table 3.1: Comparison of OctoUML with some other tools mentioned in the *Related Work* section

Tool	**Notation : Informal(IF)/Formal(F)**	**Recognition**	**Multi-touch**
Flexisketch	**IF** (hand-drawn)	predict symbols based on incremental learning	N/A
SUMLOW	**IF** and **F**, but not simultaneously	holistic recognition	N/A
Knight	**IF** and **F**, but not simultaneously	eager recognition	N/A
Calico	**IF** (hand-drawn)	beautification of shapes	N/A
OctoUML	mix of **IF** and **F** (simultaneously)	selective recognition	enabled

3.3.1 Informality and Formality

Based on studies done by [49, 51, 70], we report some common practice behaviours that occur during software modelling meetings:

- Designers combine informal and formal models.
- Designers often alternate between CASE tools and whiteboards.
- Designers prefer all-purpose sketches which refer to many scenarios over sketches dedicated to a single scenario.
- Sketches rarely follow notational convention.
- Sketches are used at different levels of abstraction.
- Designers sketch different types of diagrams with different perspectives.
- Designers extend formal notations in order to explain their ideas to others.

Whiteboards support informal design by ensuring the users a total freedom in creating and using a variety of modelling notations. For example, informal hand-drawn sketches can be used to express abstract ideas representationally, allow checking the entirety and the internal consistency of an idea as well as facilitate the development of new ideas [69]. Informal sketches, as well as various informal tools are used by software developers during their work activities [51,54,75]. However, some tasks like model transfer and persistence are difficult and require a redundant work by re-drawing the design solution using a Computer-Aided Software Engineering (CASE) tool. CASE-tools provide a limited set of modelling notations, hence restrict designers' expressiveness by imposing the notation that can be used. Modelling tools should be holistic in order to support software designer's imagination and creativity. To that end, our tool allows a simultaneous creation of both informal and formal notations on the same canvas. The informal notations can be created using free-hand sketches, while the formal notations can be either hand-drawn following a specific syntax or created using computer-drawn ready-to-use elements available in the menu. At the moment, and for the creation of formal elements, our tool

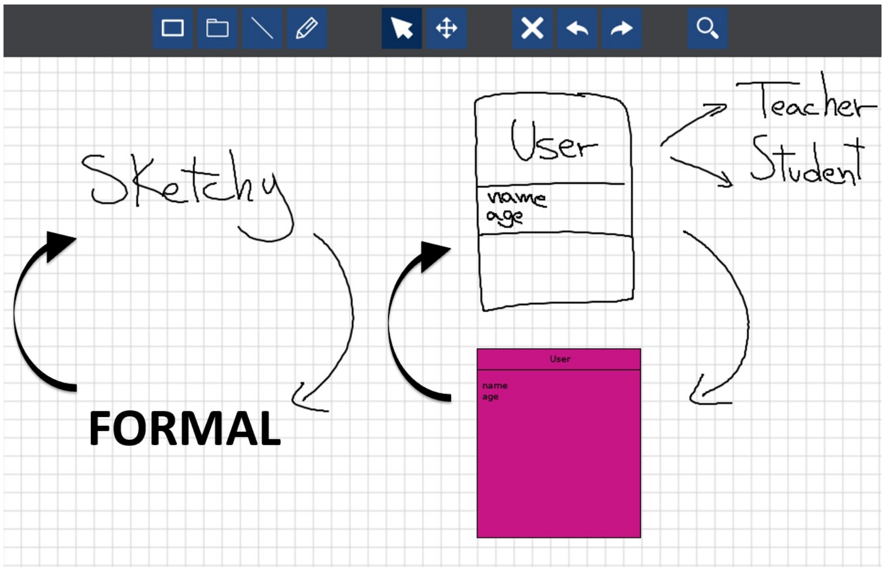


Figure 3.1: Combination of different notations on the same canvas

mainly supports UML class diagrams, but in the future we aim to support other types of UML diagrams. Figure 3.1 illustrates the main canvas of our tool. It shows how our tool allows the combination of informal and formal notations on the same canvas. Moreover, it shows how designers can transform the notations from one state to another i.e. from informal to formal and vice versa.

3.3.2 Recognition

Walny et al. [75] demonstrated that sketches have a life-cycle. In particular, a sketch starts as an informal representation of one idea and later on ends up having a formal representation. To facilitate that process as well as support tasks like: model transfer to third-party CASE tools, code generation and model documentation, our tool supports the transformation of UML hand-drawn elements to formalized computer-drawings and vice versa at any time during the modelling sessions. This has been made available using *PaleoSketch*, a primitive sketch recognition system [76]. There are two aspects that favor the flexibility and elasticity of the recognition process. Firstly, we allow users to select what they want to recognize in advance. Secondly, users can use undo/redo commands in order to move easily between the two forms; sketchy and formal.

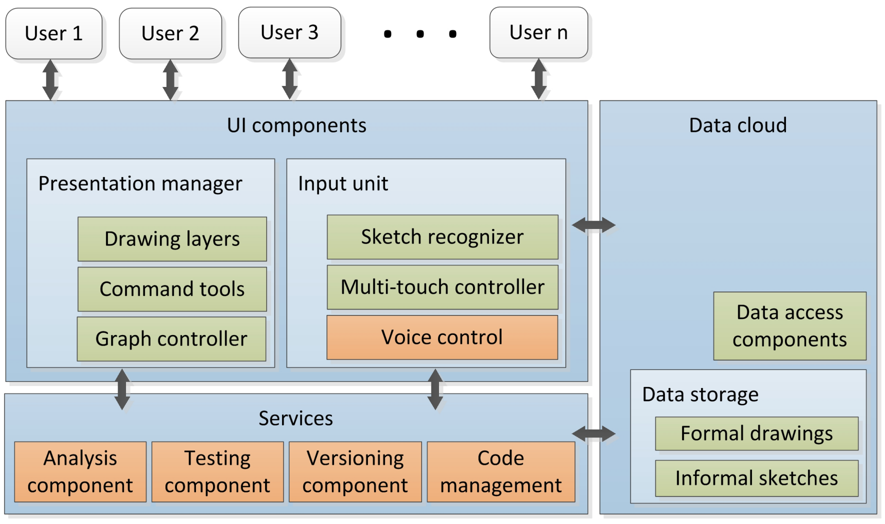
3.3.3 Layering and Multi-touch

Having been inspired by the recent version of the Altova UModel tool [2], we decided to equip our tool with a layering mechanism. In particular, the software design is part of one layer which we call the formal layer. While another layer, the informal layer, contains the informal sketchy elements e.g. hand-written comments, illustrative drawings, highlighting arrows or circles, etc. The user can then select to see combined layers or layers in isolation. A key advantage of such layers is that they allow the isolation of informal and formal elements. As a consequence, designers will be able to move and edit the content of each layer independently without disturbing the rest of the design. For instance, users might want to archive, print or share the formal designs without including the sketchy elements. In that case, the formal layer can be a solution for them. On the other hand, having the two layers combined could help reveal some existing ambiguities in diagrams as well as give more insights to increase one's understanding of concepts, mainly, during diagram reviewing cycles. Baltes and Diehl stated that quite often two or more people are involved in sketching when the whiteboard is used as a medium [10]. We enabled our tool to support multi-touch. Multi-touch is an interaction technique that permits the manipulation of graphical entities with several fingers at the same time. This option allows concurrent collaborative modelling. In particular, it enables two or more designers to simultaneously work on the same canvas of the same device, especially when the device is an interactive whiteboard or a large touch screen.

3.3.4 Other features

CASE tools are better than whiteboards when we consider some aspects such as undo/redo and re-sizing utilities. For that, we enabled our tool to support the aforementioned features in order to allow designers to easily correct mistakes and have liberty to change the size of the elements. Sometimes designers complain about the limited size of tools' modelling space or canvas which may not be enough to capture all their design ideas. To overcome this inconvenience, the drawing canvas of our tool supports panning and zooming in/out actions. Panning allows users to drag the canvas in all directions in order to find more space for their designs. In addition, zooming helps to change the scale of the canvas, hence to enhance the visibility and readability of the designs.

[2]http://www.altova.com/umodel.html



(currently implemented components are presented in green)

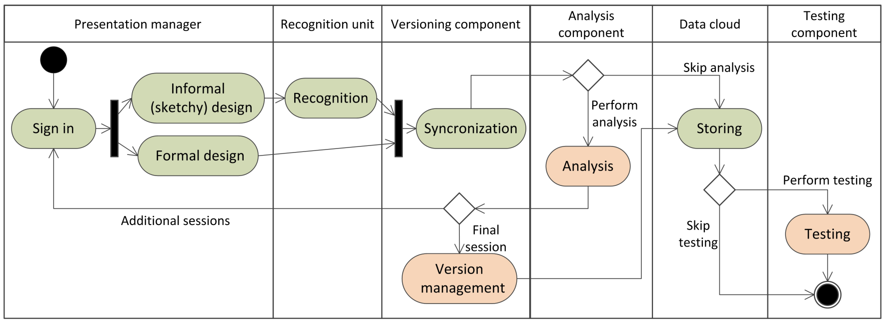
Figure 3.2: Architectural Components of OctoUML

3.4 Design

In this section we present the current architecture of OctoUML. The architecture is organized in a way to effectively fit with complex business work-flows, data, and security needs as well as to allow for future integration of different modules and other enterprise applications.

The key architectural components of OctoUML are presented in Figure 3.2. The environment contains three major components: *UI component*, *Data cloud* and *Services*. The current version of the system offers only the UI and data cloud components. Additional services will be added during future development.

UI component consists of two separated but interconnected parts: *Presentation manager* and *Input unit*. The *Presentation manager* provides means for performing stylus or touch-based input commands on devices being used. *Drawing layers* include support for both informal and formal modelling layers. Depending on the chosen layer, users are presented with an appropriate toolbar. The *Command tools* are responsible for transferring the inputs from users to different controllers. The *Graph controller* allows switching between different input techniques as well as combining of different layers. The *Input unit* is responsible for processing different inputs. In particular, a *Sketch recognizer* is



(currently supported activities are presented in green)

Figure 3.3: The Modelling Process

provided to recognize and transform informal models into formal concepts, and hence allows to maintain and transfer the designs for further processing tasks. A *Multi-touch controller* captures and coordinates the inputs from different touch-points. All the program data are saved and stored in the *Data cloud*. Our tool uses a set of data structures to manage and maintain the sketched elements, formalized designs, and session control for users. The modelling process and dynamic aspects of the system are presented in Figure 3.3.

3.5 Evaluation

In order to answer the research questions presented in Section 1, we prepared user studies. Sixteen subjects were engaged in a design assignment. The assignment was to create a UML class diagram of a given scenario using our tool. To give a global overview of the subjective assessment of our tool's usability, we asked our subjects to answer the System Usability Scale (SUS) questionnaire [77]. Furthermore, we planned semi-structured interviews using both closed and open questions. The interviews were held after the completion of the design assignment. Our main concern was to get feedback from the participants regarding their experience in using OctoUML, so we focused on qualitative data more than quantitative data. We followed the grounded theory methodology [78], and used *NVivo*[3] in the qualitative data analysis process. More details together with the results are reported in the following subsections.

[3]http://www.qsrinternational.com/

3.5.1 Participants and Modelling Expertise

Sixteen software engineering students and researchers were involved in doing the assignment and subsequently the interviews. In particular, there were four master students, ten PhD students and two post-doctoral researchers. Five participants have an experience in industry for some period of time. Twelve people worked on the design task in pairs, while the rest worked individually. Overall, the participants are experts in software modelling and have some experience in using UML. In fact, nine participants claimed to have high expertise in software modelling, five have a moderate expertise, while only two have low expertise. All participants believe that software design is a critical task for successful software development and evolution. In previous occasions, all but one participant did software modelling with other people in teams (collaborative modelling). The participants have a practical experience with a variety of modelling tools. These tools range from whiteboards, pen&paper to CASE tools like *Enterprise Architect*, *Visual Paradigm*, *Dia*, *ArgoUML* and *Papyrus.*

3.5.2 Design Task

We formulated a simple design problem, and asked the participants to design a domain model class diagram solution of it using OctoUML. Before starting with the task, we gave the participants a brief introduction regarding the features and functionalities of our tool. We directly observed the design processes and took notes. When the participants asked some specific questions about the design assignment, we told them that it is up to their interpretation. They could, however, ask questions concerning the design environment e.g. how to use certain tools. The participants did not get any help unless they asked for it. An interactive whiteboard was chosen as an input medium, and the design sessions were video-recorded. The text of the design assignment is presented in the following paragraph.

E-Learning System. The system is used by teachers, students and an administrator (who is also a teacher). One teacher is responsible for many courses. Consider that courses consist of many topics. Students can enroll into different courses. There is a news section within the system. Teachers add news for a specific course and the students can read them. Every course ends with an evaluation test. Teachers create a test and the students have to do it. The students get one of these grades: fail, pass, good, or very good.

3.5.2.1 Design Task Observations

While carrying out the design task, the participants were observed in order to understand their activities. This was done in two different ways: First, we directly observed the behaviour of the participants as they were performing the task and took notes. Second, the participants were recorded via a digital video-camera. This let us observe their behaviour indirectly through records of the task. The notes which we took were expanded by transcribing elements of the video recordings. At the beginning of the task the participants spent around one minute reading the assignment, then they proceeded with designing the solution. While carrying out the design task, some participants first created many UML classes then they associated them with different kind of relations. Other participants followed another strategy by creating two classes in the first place, then, they defined the relation between the two classes before continuing to create other UML classes. Even when two people were working on the same design at the same time, they rarely interacted with the e-whiteboard at the exact same time. Most of the participants tended to create the classes sequentially, and they discussed the properties of one class before proceeding with the creation of another class. Furthermore, they often divided the work between themselves, e.g. one was interacting with the tool to create classes, and the other one was reading the assignment as well as providing ideas for the solution. The participants were given a brief introduction about functionalities of the tool. Nevertheless, during the design task, some of the participants were confused and hesitant about using some features being provided by our tool. This was actually observable at the beginning of the task, but seemed to be overcome later on. In fact, the participants became more confident as they were gradually and progressively interacting with the tool. Moreover repeating some actions, such as selection and creation of classes, let them in some manner experience our environment' functionalities.

3.5.3 SUS Questionnaire

The *System Usability Scale* is an easy, standard way of evaluating the usability of a system [77]. It is a form containing ten statements, and users provide their feedback on a 5-point scale (1 is strongly disagree and 5 is strongly agree). It effectively differentiates between usable and unusable systems by giving a measure of the perceived usability of a system. It can be used on small sample sizes and be fairly confident of getting a good usability assessment [79]. The participants were given the forms directly after they finished with the design task. We considered the SUS score as a “preliminary feedback” on the usability of our tool. However, in order to consolidate the current findings, more people will be involved in testing our tool and answering the SUS questionnaire.

3.5.3.1 SUS Result

All sixteen participants filled out the SUS questionnaire. We calculated the SUS score reported by each participant. After that, we calculated the average of the usability values of all participants to obtain the overall OctoUML usability score[4]. The lowest score was 65 and the highest was 95, with an average score of 78.75. It falls at around the 80th percentile, and would result in a grade of a "B" which is a high usability score according to [43].

3.5.4 Interviews

After answering the SUS questionnaire, each participant was involved in a *semi-structured* interview. The conversations were recorded using a digital voice recorder. The interviewers took some notes which were expanded afterwards by transcribing the audio recordings. Both closed and open questions were used, and respondents' answers were quantitatively and qualitatively analyzed.

3.5.4.1 Interviews' Results

Several threads run through the interviews. Such threads are categorized as themes and reported subsequently.

A. Tool Usability

Ease of Use. All participants pointed out that our tool is intuitive, simple and easy to use.

> "*Easy to get started with, I do not have to understand the UML-standard*", "*It's simple to use*", "*Easy to change things*"

Learn-ability. The participants also stated that the tool is easy to understand and learn.

> "*easy to understand*", "*Easy to grasp what is what*", "*Intuitive for the most part*", "*So easy to learn*"

Efficiency. We let our environment inherit the fluidity and immediacy of a standard whiteboard. Furthermore, we wanted to maintain the recognition process to be as smooth and fast as possible. Some participants were impressed by the fluidity and immediacy of our tool in drawing and creating notations as well as in recognizing the sketchy elements.

> "*Very quick to draw classes and associations compared to CASE tools*", "*Very smooth and quick recognition*"

Satisfaction. The basic functions of our tool met the expectations of most of the participants. Overall, the participants liked our tool. Two participant highlighted the eligibility of our tool in collaborative team modelling. One participants stated that he likes the "selective recognition" mechanism.

[4]https://goo.gl/uwlwIp

> "*Very straightforward once you get used to it*", "*It is really good for team design*", "*Nice that you select what to recognize*"

However, some challenges in tool usability were identified. one participant asked for more flexible switching between *informal* and *formal* input modes .

> "*I did not like how I switch between input modes*"

Some of the participants did not like the manner by which the elements are being selected i.e. by activating a dedicated button.

> "*Should select by just clicking on element without selecting the selection tool first*"

One participant stated that "typing-in" classes name and their properties using the virtual keyboard was inconvenient task due to the time that it takes, and asked us to find a better solution.

> "*Typing-in is a bit slow*"

We previously mentioned that our tool mainly supports UML class diagrams when creating formal "computer-drawn" elements. Some participants expressed the need of being able to create other types of UML diagrams as well as having more tool options.

> "*I want more features, options, more types of diagrams*"

We consider those challenges, hence they will be used together with the overall feedback of the users as a basis for future improvement of our tool.

B. Informal and Formal Notation

There was a strong belief among the participants (12 people agreed, 2 were undecided and 2 disagreed) that informal notations (i.e. sketches) support the formal design. Sketches can be used to interconnect different components

> "*define the components involved and the interactions among them*".

Sketches are valuable artefacts beyond being just explorative drawings

> "*obtain a formal document, not just a sketch*".

There was also a strong belief among participants (11 people agreed, 2 were undecided and 3 disagreed) that having sketches beside the formal design can allow for a better expression of ideas

> "*map out the domain of the functionality and make sure everyhting is been covered*", "*get an idea of how things fit together*".

Eleven people claimed that sketches can enhance the understandability and readability of the formal designs

> "*it is mostly for myself to get a clear understanding*", "*it is helpful to make you understand what you are gonna build*"

On the other hand, one person said that having sketches beside the formal design could complicate the diagram

> "*No, sketching will introduce complexity*".

Half of the participants think that being able to sketch beside creating formal designs in one tool can replace the need of sketching on a whiteboard or a paper.

> "*Yes, you could perform all the functions that you can do on standard whiteboard. Be able to quickly show ideas, and you do not have to take pictures of the whiteboard*".

While some participants claim that still people will not stop using standard whiteboards as well as pen and paper. The dependency on such tools, as the participants argued, arises from their immediacy and ease of use as well as being at easy disposal.

> "*No, because you will never remove the paper from offices*", "*right now still very dependent on paper*"

Sketching down thoughts and ideas on paper or whiteboard when designing software is a common behaviour when designing software. In fact, all participants do sketches to some degree. Some participants mentioned that it depends on the complexity of the problem

> "*I do not sketch when it comes to simple stuff*".

According to the participants' responses, the main purposes of sketching are to:

[a] understand problems and start to explore solutions

> "*to define the components involved and the interactions among them*", "*start putting the solution together*"

[b] brainstorm as well as explore ideas

> "*to brainstorm*", "*to facilitate how to express our ideas*", "*to visualise what is in my head*".

[c] communicate and discuss their ideas

> "*give an explanation for other people about the system or a specific problem*", "*communication of ideas in teams*".

3.6 Discussion

Before starting with the design task, the participants were given a short introduction about our environment and its functions. However, while we were observing the participants creating their designs, we noticed that they were not very inclined to use the sketching feature which was illustrated during the short introduction. In fact, only six (out of sixteen) participants used the sketching feature. We think that the time of the introduction part was most likely not enough to make the participants feel comfortable in using free-hand sketches. Furthermore, the design problem that we have chosen for the design task was simple and easy to solve. We tried to simplify it as much as possible to make it solvable in a short time. We believe that the simplicity of the design task could have defeated the need of sketching it up. The participants could easily get a good grasp of the design task simply by reading it. However when we did the interviews, the majority of the participants did agree that being able to mix informal and formal notations could support the design process and flow, thus could bridge the gap between the process of prototyping ideas on a paper and the process of entering a formalized version of such ideas in a CASE-tool. Next, we discuss the research questions based on our interpretation of the results.

- RQ.1 Does our tool provide a usable environment considering issues like ease of use, efficiency and user satisfaction?

OctoUML is designed to offer a usable interactive environment. First we focused on understanding some common practice activities that occur during software modelling sessions. Then, we tried to consider and adopt some novel interaction modalities which could interactively support the design process. The interviews' results show that the current version of our tool is easy to learn, effective to use, and provides an enjoyable user experience. Moreover, the results that we got from the SUS questionnaire on the usability of our tool consolidate the previous findings. Of course these results hold only for the device which is used as an input medium, the interactive whiteboard. Other media like tablets and standard PCs have different characteristics. Tablets have a smaller interaction interface when compared to interactive whiteboards, and this could raise some usability challenges. To assess these challenges further tests on different input media are required. On the other hand, our tests revealed some usability challenges related to our system. The main issue is the selection tool. The participants had to click on a specific button to activate the selection mode. According to some participants, that was not a user-friendly choice. Moreover, we asked the participants for their opinion about some new interaction techniques that could be adopted by our environment in the future. These techniques were appreciated by the participants and are discussed in the conclusion and future work section.

- RQ.2 Does support for mixing informal and formal notation better support the software design process?

In practice, software systems are becoming more and more complex, and the design of complex systems needs more effort and hence more sophisticated designing tools. In such cases, having the possibility to use informal notations beside the formal ones can better support designers' activities in understanding the problems, exploring solutions, brainstorming and communicating ideas. This is in line with the study of Mangano et al. [70]. They stated that informal notations, i.e. sketches, allow software designers to discuss design alternatives as well as mentally simulate the behaviour of complex systems.

Apart from the fact that our subjects did not sketch informal elements frequently, the majority of them think that being able to simultaneously create sketches and formal designs in one design environment could support the design process. Indeed, software designers often sketch their early-design ideas on a paper. However, when they want to preserve the design, they do a redundant work by re-drawing the solution using CASE-tools. Furthermore, they may forget to include some sketched ideas in the design formalization process.

There was a strong belief among the interviewees that having the possibility to create informal elements, i.e. sketches, assists the process of ideas expression and enhance the understandability of formal designs. The informal notations can be used both in brainstorming sessions and while creating the formal design. In the former case, they are used for design exploration and can be volatile. While in the latter case, the informal elements could be used to sustain and describe a specific design problem as well as support the formal design in conveying and reinforcing the information that they carry. Informal sketches, for example, may have a very close mapping to the problem domain. As a result, they could be valuable artefacts beyond being just explorative means.

3.7 Threats to Validity

Construct Validity. The design task was simple, specific and easy to do. Moreover, it is relatively small compared to real world design problems. This might limit the creativity of the designers as well as influence the amount of discussions and usability interactions. However, during the interviews, we asked the participants to give their general opinion about the efficiency of our tool when it comes to handling different design problems that vary in size and complexity.

Internal Validity. None of our subjects was familiar with our tool and its functions. To mitigate this, we gave the participants a short introduction explaining the features of the tool. Moreover, during the interviews, the participants might want to please the interviewers by giving them a positive feedback. To mitigate this, we asked the participants to answer the SUS questionnaire which allowed them to give feedback anonymously.

External Validity. The participants being involved in both the design task and the interviews may not represent the general population of software designers. This could threaten the generality of the results. However we involved people with different backgrounds, modelling expertise and academical degrees.

3.8 Conclusion and Future Work

Currently, most CASE-tools are modelling (or even diagramming) tools. Indeed, they lack support for the majority of design activities in which developers are engaged. In this paper, we presented a proof of concept of a new generation software design environment. Basically, our tool allows for simultaneous creation of both informal freehand sketches and formal computer-drawn notations. The users of OctoUML can create software designs by performing simple intuitive touch gestures. Moreover, they can manipulate the graphical entities with several fingers at the same time thanks to the *multi-touch* technology being adopted by our system. Furthermore, OctoUML supports the transformation of models from informal to formal at any time during the design sessions. We evaluated our tool by conducting user studies. The results show that OctoUML, as perceived by our subjects, provides a usable environment in terms of ease of use, efficiency and user satisfaction. Moreover, it seems that giving the possibility to create informal and formal notations in one software design environment could support both the design process and its flow.

Future Work. We will continue to realize our vision [71] of a new generation of software design environment. OctoUML will be equipped with microphones to record the spoken discussions, and a recognition system will be provided to interpret users' voice commands. To open up new opportunities for remote interactive collaborative design, our tool will be enabled to support remote collaborative sessions between geographically distributed teams as well as in class room environment between students and teachers. We also aim to integrate OctoUML with other software engineering tools to provide effective support for different development tasks (e.g. requirements gathering, testing, coding and versioning) and analysis tasks (e.g. performance).

Chapter 4

Paper C

Using Voice Commands for UML Modelling Support on Interactive Whiteboards: Insights and Experiences

R. Jolak, B. Vesin, M.R.V. Chaudron

In Proceedings of the 20th Ibero American Conference on Software Engineering (CibSE) @ICSE17, pp. in print. 2017.

Abstract

The ultimate goal of software design tools is to provide efficient support for designing software systems. In our previous work we presented OctoUML, a prototype of a new generation software design environment. OctoUML supports collaborative software design and enables different input methods for creation of software models at various levels of formality. Recently, we added a voice recognition unit into the system and provided users with possibilities of giving voice commands. This paper presents insights and gathers experiences from enabling a voice interaction modality within software design environments. By conducting two user studies, we found that (i) the general perception regarding the usability is positive, (ii) the voice interaction modality is mostly preferred for text input, and (iii) the employment of voice interaction modality within the software design environments enhances the efficiency of the software design process.

Keywords: MDE; software design environments; interaction modalities; voice recognition; accessibility; integration with HCI.

4.1 Introduction

The Unified Modelling Language (UML) is a well-known and commonly used language for software design [18, 80]. Various Computer-Aided Software Engineering (CASE) tools can help software developers to create a UML-model of software. Although these tools can be run today on different devices (e.g. computers, tablets, etc.), the input methods that they support are still limited, and therefore diminishing their usability [81]. Moreover, the interaction with such tools is not always well-designed for user experience, easy learning and effective use. Consequently, many software designers tend to avoid using CASE tools which are considered complex and time-consuming [6, 82].

Multimodal systems use integrated multiple interaction modalities (e.g. sketch, touch, voice, etc.). Oviatt and Cohen [13] illustrated the importance of multimodal systems in reshaping daily computing tasks and predicted their future role in shifting the balance of human-computer interaction much closer to the human. Harris [83] highlighted the importance of voice, as an incorporated modality within multi-modal interfaces, in both opening up a new cognitive dimension and overcoming the barriers of "era-ending" graphical interfaces. The usability of software design environments could be enhanced by introducing voice recognition, as this could terminate the need of using the keyboards and counteract the matter of menus hierarchy. Yet, the effectiveness could be also improved even further [84].

Interactive whiteboards support touch-input and are useful tools for creating knowledge and sharing information. They increase the learning outcome for students in the classroom [85, 86] and enhance the effectiveness and interaction at company meetings [87]. Such capabilities of interactive whiteboards make them potential input devices for software design tools.

In a previous work, we presented our vision on a new generation of software design environments [71]. One of the characteristics which we proposed for such environments is that they should be equipped with microphones to record the spoken discussions, and have a recognition component to interpret users' voice-commands.

Recently, we created a software design environment, called *OctoUML* [88], that provides efficient support for the design of hardware and software architectures. OctoUML can be run using different input devices ranging from desktop computers, over touch screens, to large electronic whiteboards. Beside supporting the creation of software models at different levels of formality, OctoUML is equipped with tools for multi-touch and sketch recognition which enable concurrent collaborative modelling and sketch formalization, respectively. Our current goal is to improve the usability and efficiency of OctoUML by supporting and evaluating new modes of interaction. To this end, we added a voice recognition component into OctoUML. This component is capable of accommodating the most commonly used functions of the system (watch the demo videos[1] [2]).

This paper presents insights and gathers experiences from supporting voice interaction modality within software design environments. The contribution of this work is mainly based on answering and discussing the following three research questions:

- **(RQ1)** For which features of a software design environment do users find it practical to interact through voice commands?
- **(RQ2)** What are the perceptions of users regarding the usability of the voice interaction modality supported by OctoUML?
- **(RQ3)** Does the employment of voice interaction modality within the software design environments enhance the efficiency of the software design process?

To answer these questions, we conducted two user studies. In the first user study, *USS1*, a version of OctoUML without the voice recognition feature was used (*OctoUML-Lite*), while for the second user study, *USS2*, we enabled the voice recognition feature (*OctoUML-V*). Details regarding the two user studies are presented in Section 4.4.

4.2 Related work

A single interaction modality does not allow for an effective execution of tasks and use of environments [89]. Multimodal systems provide possibilities to use a combination of modalities or change to a better-suited modality, depending on the specifics of the task [90].

Like [90], we believe that supporting multiple interaction modalities within software design environments would make the interaction: (i) more intuitive, especially during software design processes when designers discuss and communicate their ideas to each other via voice and gestures, and (ii) more effective by allowing the users to switch to a better-suited modality for the execution of one particular task. Moreover like [13], we believe that multi-modal systems have the potential to shift the balance of human-computer interaction much closer to the human.

Repetitive Strain Injury (RSI) is a condition where pain and other symptoms occur in muscles, nerves and tendons after doing repetitive tasks (e.g. using a keyboard frequently) [91]. Several studies have been carried out to help programmers with RSI by using voice recognition techniques [92–94]. However,

[1]http://goo.gl/JP4Mfg
[2]https://goo.gl/uU6Zm3

Mills et al. [95] pointed out that such techniques have a high rate of faults and errors which negatively influences their usability.

Lahtinen and Peltonen [81] presented an approach to build speech interfaces to UML tools. The authors set up a spoken language to manipulate the UML models, and built a prototype (called *VoCoTo*) of a speech control system integrated with a CASE-tool (*Rational Rose*[3]). They stated that speech recognition is applicable to be used to enhance the interaction with UML tools. Soares et al. [96] presented a framework, *VoiceToModel*. It allows the creation of requirements models, e.g. conceptual UML class diagram, from speech recognition mechanisms. They evaluated their prototype through an experiment with fourteen computer science students. The users experienced some difficulties in using *VoiceToModel*. However, they were overall satisfied and liked the interaction model provided by the framework.

The objectives of the two aforementioned related works [81, 96] were to assess the viability of their systems and get a *proof of concept* instead of making a statistical proof for their approaches. In this study, we aim to discover which tasks in the software design process are better qualified to be supported by a voice interaction modality. Moreover, we want to assess the efficiency of our approach by comparing two versions of OctoUML; one version is enabled to recognize voice-commands and the other one is not.

Nishimoto et al. [97] designed a multi-modal generic drawing tool that supports speech, mouse, and keyboard inputs. Their tool has a speech recognition system based on Hidden Markov Models [98]. The evaluation showed that their multi-modal drawing tool with speech recognition reduces the required operation time per task, mouse movement, and commands number.

We assess the efficiency of the employment of the voice interaction modality in software design environments by following a similar evaluation approach to [97]. In particular, we measure the amount of time and number of steps that are required to accomplish a specific task using two different input modalities (keyboard/touch and voice) that are supported by OctoUML. Furthermore, we run formative evaluations in order to get feedback on the usability and usefulness of our approach as well as get suggestions for improvement.

4.3 The Design Environment: OctoUML

OctoUML was mainly designed to bridge the gap between early software design process, when informal tools (e.g. whiteboards) are typically used for brainstorming and design reasoning, and subsequent formalization process, when formal tools (e.g. CASE-tools) are used for documentation purposes. Such a gap or discontinuity was reported by Budgen [4] in his study on why

[3]http://www-03.ibm.com/software/products/en/rosemod

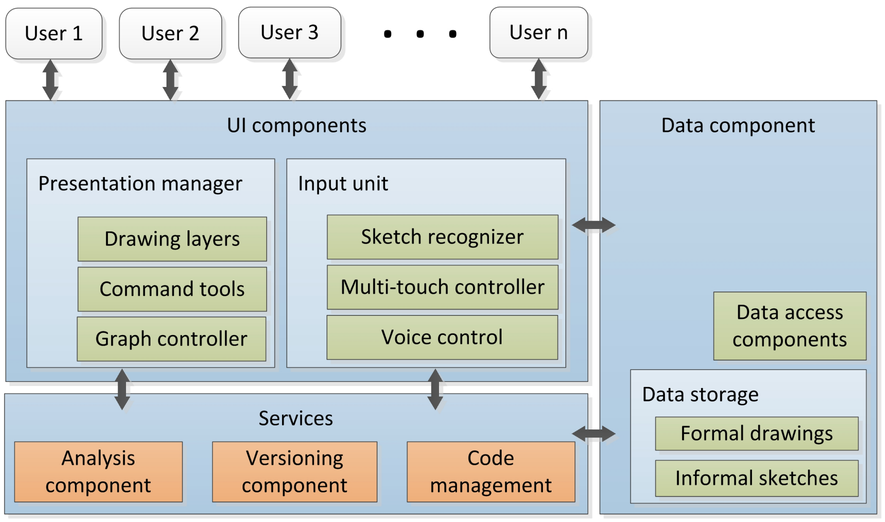


Figure 4.1: Architecture of OctoUML (Currently implemented components are presented in green.)

software design environments do not support realistic design practices. Indeed, ideas and logical basis for the design solution can be easily lost when moving from the early design reasoning process to the formalization process, especially when there is a non-short timespan between the two processes.

OctoUML is a collaborative software design environment that allows the creation and modification of diagrams at different levels of formality. In particular, the environment allows mixing of both informal (e.g. sketches) and formal (e.g. UML class diagrm) modelling notations. Furthermore, OctoUML is equipped with tools for multi-touch and sketch recognition, and supports the transition from informal notations to formal ones. The users of OctoUML are provided with options for moving, resizing, grouping and separating software elements, regardless of their informal or formal character [72].

Figure 4.1 shows the architecture of OctoUML. The architecture is organized in a way to effectively fit with complex business work-flows as well as to support future integration of different modules and other enterprise applications.

The design environment contains three major components: *UI component*, *Data cloud* and *Services*. The current version of the system offers only the *UI* and *Data cloud* components. Additional services will be added during future developments. The *UI component* consists of two separated but interconnected parts: *Presentation manager* and *Input unit*. The *Presentation manager* provides means for performing stylus or touch-based input commands on devices being used. *Drawing layers* include support for both informal and formal modelling styles. Depending on the chosen layer, users are presented with an appropriate toolbar. The *Command tools* are responsible for transferring the inputs from users to different controllers. The *Graph controller* allows switching between different input techniques as well as combining of different layers. The *Input unit* is responsible for processing different inputs. In particular, a *Sketch recognizer* is provided to recognize and transform informal models into formal concepts, hence allows to maintain and transfer the designs for further processing tasks. A *Multi-touch controller* captures and coordinates the inputs from different touch-points. Sketched elements as well as formalized designs are saved and stored in the *Data cloud*. The *Voice control* component is the main focus of this paper and is described in the following subsection.

4.3.1 Integration of The Voice Control Component

In order to improve the usability of OctoUML and increase its accessibility, we integrated a voice-commands control component within the *Input unit*. The component is capable of handling the most commonly used functions during the design process. Thus, users can use voice commands in order to create and manage various elements of software diagrams. The *Voice control* component was implemented using *Sphinx4*. Sphinx4 is an open source voice recognition library developed by Carnegie Mellon University [99].

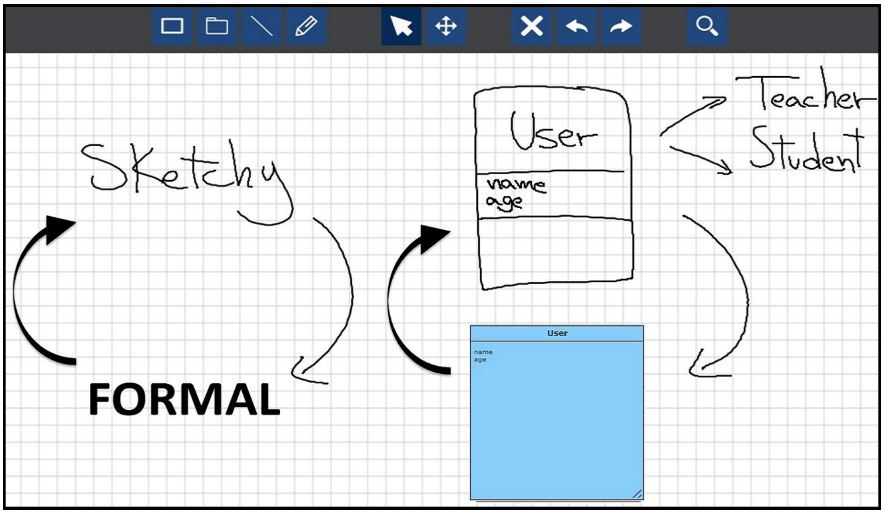


Figure 4.2: The main canvas of OctoUML.

There are two main types of commands that trigger the *Voice control* component, and therefore allow the interaction with OctoUML via voice:

- Type α: activation/execution of the different interaction tools available on the top of the main canvas (see Figure 4.2) such as create class, select, undo/redo, etc.
- Type β: assign names to the created packages and classes.

For the latter type of commands (β), the current version of the voice control component is based on a predefined dictionary that contains a list of expected words to be used. Table 4.1 provides more details on the voice commands of types α and β.

Table 4.1: Different types of voice commands.

Type α	
Command	*Description*
Create Class	selects the class drawing tool
Create Package	selects the package drawing tool
Create Edge	selects the edge drawing tool
Selection Mode	selects the select tool
Moving Mode	selects the moving/panning tool
Undo/Redo	undo/redo actions
Type β	
Command	*Description*
Name	give names to classes and packages. *For instance*: double tap on one class, then say “Name” followed by the desired name.

4.4 Study

We conducted two user studies to answer the research questions that are presented in Section 4.1. The two studies were conducted in two different sessions. In the first user study, *USS1*, a version of OctoUML without the voice recognition feature was used (*OctoUML-Lite*), while in the second user study, *USS2*, we enabled the voice recognition feature (*OctoUML-V*). The purpose of carrying out the user studies was to gather experiences, identify faults and discover areas of improvement of OctoUML. Furthermore, we wanted to get and subsequently compare quantitative data regarding the time and steps that are required for the accomplishment of a specific task using the two versions of OctoUML.

USS1 involved fourteen software engineering students (ten PhD and four M.Sc. students) and two post-doctorate software engineering researchers, whereas *USS2* involved fourteen participants (three PhD and eleven M.Sc. software engineering students). All the subjects are familiar with software design and have experience in using the UML. Moreover, they think that software modelling is a critical task for successful software development and evolution (See Figure 4.3: the range of ratings is [1 to 5] where 1 is the most negative score and 5 is the most positive score).

The subjects have a practical experience with a variety of modelling and design tools. These tools range from whiteboards, pen&paper to CASE-tools like Enterprise Architect[4], Visual Paradigm[5], Dia[6], ArgoUML[7] and Papyrus[8].

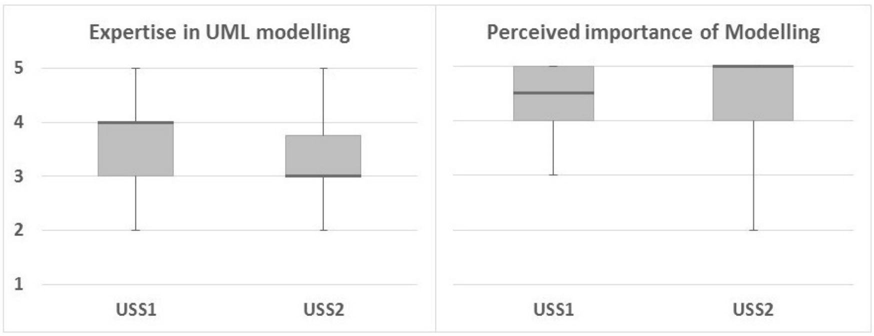


Figure 4.3: Expertise in UML modelling and perceived importance of modelling.

We deployed the two versions of our tool *OctoUML-Lite* and *OctoUML-V* on a multi-touch interactive whiteboard (78 1/2" W x 53 3/8" H) that was connected to a Windows 7 PC with a Core Duo 3.00 GHz processor. Later on, each subject was introduced to the functionalities of OctoUML. The introduction lasted around 10 minutes on average. After that, the subjects were given a *software design assignment* which consisted of a short text describing a system to be designed using UML class diagram. The text of the assignment is provided in the following paragraph:

E-Learning System. The system is used by teachers, students and an administrator (who is also a teacher). One teacher is responsible for many courses. The courses may consist of many topics. Students can enroll into

[4]http://www.sparxsystems.com/products/ea/
[5]https://www.visual-paradigm.com/
[6]http://dia-installer.de/shapes/UML/index.html.en
[7]http://argouml.tigris.org/
[8]https://eclipse.org/papyrus/

different courses. There is a news section within the system. Teachers add news for a specific course and the students can read them. Every course ends with an evaluation test. Teachers create a test and the students have to do it. The students get one of these grades: fail, pass, good, or very good.

The subjects involved in USS2 were also given a sheet of paper containing a list of detailed voice commands. While carrying out the design assignment, the subjects were observed and video-recorded using a digital video camera in order to note and understand their activities. On average, each subject completed the design assignment in 20 minutes. After the completion of the design assignment, we asked the subjects to answer a System Usability Scale (SUS) questionnaire [77] in order to give a global overview of the subjective assessment of the usability of the two versions of OctoUML. SUS can be used on small sample sizes and be fairly confident of getting a good usability assessment [79]. After answering the SUS questionnaire, each participant was involved in a semi-structured interview. The conversations were recorded using a digital voice recorder. The interviewers took some notes which were expanded afterwards by transcribing the audio recordings, and the data were quantitatively and qualitatively analysed.

4.5 Results

The results are presented in a form of answers to the research questions that we posed and reported previously in Section 1.

R.Q.1. *For which features of a software design environment do users find it practical to interact through voice commands?*

We asked the subjects who were involved in the user study USS2 to rate the eligibility (suitability) of the supported voice commands of type α (create class, create package, create edge, selection mode, moving mode and undo/redo) and β (name class and name package), together with two additional commands that could be supported in the future: (i) *add attribute/method*, to allow the creation of attributes and methods via voice commands; and (ii) *delete class*, to allow the deletion of selected classes. The results are presented in Table 4.2, where the scale that is used for the rating ranges from 1 (not important) to 5 (very important).

We also asked our subjects if there are any other functionalities that are desired to be supported by voice commands. The following list reports these desired functionalities, where every functionality was mentioned by at least one subject:

- naming and changing the type of association,
- save, open, import and export files,

Table 4.2: Suitability (eligibility) of different types of voice commands.

Voice Commands	**Results**				
	User Study	Median	1st Quartile	3rd Quartile	Inter-Quartile Range
Type α (Creation)	USS2	4.00	3.00	5.00	2.00
Type α (Selection)	USS2	4.00	2.25	5.00	2.75
Type α (Moving)	USS2	4.00	3.00	4.00	1.00
Type α (Undo/Redo)	USS2	4.50	3.25	5.00	1.75
Type α (all)	USS2	4.00	3.00	5.00	2.00
Type β	USS2	4.50	4.00	5.00	1.00
Other Voice Commands	**Results**				
	User Study	Median	1st Quartile	3rd Quartile	Inter-Quartile Range
Add attribute/method	USS2	4.00	4.00	5.00	1.00
Delete Class	USS2	3.00	2.00	4.00	2.00

- create a new diagram,
- re-arrange the classes and packages,
- select and deselect classes, packages or edges,
- zoom in and out,
- exit the application,
- define your own voice commands.

R.Q.2. *What are the perceptions of the users regarding the usability of the voice interaction modality supported by OctoUML?*

The perceptions regarding the usability of the two versions of OctoUML (*OctoUML-Lite* and *OctoUML-V*) were collected via: *(i)* the SUS questionnaire and *(ii)* the semi-structured interviews that were run after the completion of the design assignment. As a matter of fact, the collected perceptions reflect satisfaction, comfort and acceptability of use. The results are presented in Table 6.9. The median is indeed the same for all the measurements concerning the usability of the two versions, except the measurement of the required learning effort to get going with the system (OctoUML-V required more learning effort).

During the semi-structured interviews, various emotional responses and experiences were shared with the interviewers. The subjects enjoyed the experience of using and interacting with the software design environment, especially via voice. Perceptions regarding the simplicity and ease of use of OctoUML were positive, and the subjects valued these aspects when comparing OctoUML to other software design environments that they used previously. The experience of naming the classes in UML class diagram via voice was much more appreciated compared to the experience of using the keyboard to do the same task. OctoUML-V was perceived useful for simplifying the process of class diagram creation and recommended to people with disabilities.

Table 4.3: Perceptions regarding the usability of OctoUML-Lite & OctoUML-V

Measurement	**Results**				
	OctoUML version	Median	1st Quartile	3rd Quartile	Inter-Quartile Range
	OctoUML-Lite	4.00	3.00	4.00	1.00
Willing to use the system frequently	OctoUML-V	4.00	3.25	4.00	0.75
	OctoUML-Lite	2.00	1.00	2.00	1.00
Complexity of the system	OctoUML-V	2.00	1.00	2.00	1.00
	OctoUML-Lite	4.00	4.00	5.00	1.00
Ease of use	OctoUML-V	4.00	4.00	4.75	0.75
	OctoUML-Lite	2.00	1.00	2.00	1.00
Need of support to use the system	OctoUML-V	2.00	1.25	2.00	0.75
	OctoUML-Lite	4.00	3.00	4.00	1.00
Integrity of various functions	OctoUML-V	4.00	4.00	4.00	0.00
	OctoUML-Lite	2.00	1.00	2.25	1.25
Inconsistency in the system	OctoUML-V	2.00	1.00	2.00	1.00
	OctoUML-Lite	5.00	4.00	5.00	1.00
Intuitiveness	OctoUML-V	5.00	4.00	5.00	1.00
	OctoUML-Lite	2.00	1.00	2.25	1.25
Cumbersomeness to use	OctoUML-V	2.00	1.00	2.00	1.00
	OctoUML-Lite	4.00	3.75	5.00	1.25
Feeling confident when using the system	OctoUML-V	4.00	3.25	4.00	0.75
	OctoUML-Lite	1.50	1.00	2.00	1.00
Required learning-effort	OctoUML-V	2.00	1.25	2.00	0.75
Ease of using the Voice Interaction Modality (VIM)	OctoUML-V	4.00	3.25	4.75	1.50
Perceived effectiveness of VIM	OctoUML-V	4.00	3.00	4.00	1.00

R.Q.3.*Does the employment of voice interaction modality within the software design environments enhance the efficiency of the software design process?*

During the semi-structured interviews that were held in the user study USS2, a few subjects (5 out of 14) perceived that the voice commands of type α did not significantly enhance the design process as it was not that big of a reach or hassle for the subject to click on the buttons in order to activate/execute the tools of the software design environment. However the subjects considered this type of commands useful for people with disabilities. While the voice commands of type β were much more appreciated by the subjects who predicted their potential in replacing the use of the keyboard which was perceived a time-consuming task. For that, we only considered the voice commands of type β in the assessment of the efficiency of the employment of the voice interaction modality in the software design environment (OctoUML). To assess the efficiency, we measured the amount of time and number of steps (interactions) that are required for naming classes and packages in all UML diagrams that were created during the two user studies (USS1 and USS2) using OctoUML-Lite and OctoUML-V, respectively. The results are presented in Table 4.4. They show that the process of naming the classes and packages via voice requires the same number of steps (3 steps: *select* then *name* then *confirm*), but less amount of time with respect to the same process using the keyboard. In fact, the difference in time is significant according to Mann-Whitney's test [100] (p-value is $0.047 < 0.05$). Furthermore, the standard deviation of the naming effort for OctoUML-V (SD = 0.32) is lower than OctoUML-Lite (SD = 1.76), and indicates that the calculated times for naming class diagram elements via *voice* are more closely clustered around the mean (the variance of the required time to name different elements via voice is small).

Table 4.4: Naming effort: *keyboard* (OctoUML-Lite) vs. *voice* (OctoUML-V).

OctoUML Version	**Number of steps**	**Amount of time (Seconds)**		
		Mean	St.Dev.	Difference in mean rankings *Mann-Whitney* test (p-value)
OctoUML-Lite	3	2.12	1.76	
OctoUML-V	3	1.35	0.32	0.047

4.6 Discussion

According to our subjects, the better-suited task in software design that should be supported by voice interaction modality is when the user needs the keyboard for text input i.e. naming classes and packages via commands of type β. The main reason is that using a keyboard is not ergonomic and a time-consuming task. In fact, it is easier, faster and more comfortable to use voice instead of typing [96]. When we asked the subjects to rate the eligibility (suitability) of the different voice commands, the voice commands of type α got less suitability score than the commands of type β (α's median is 4.0 against 4.5 for β). Even if such commands are of less suitability to the subjects, their support was strongly recommended because of two reasons: (i) the interaction with the software design environment via voice is more enjoyable than using the traditional way e.g. keyboard or touch inputs, and (ii) the potential of the voice interaction modality that was perceived by the subjects in supporting people with disabilities. Indeed, some of our subjects wanted to be able to create UML class diagrams by using only voice commands. Of course for such a scenario, every possible feature and functionality of the software design environment is a candidate for voice recognition.

Overall, the perceptions regarding the usability of the two versions are similar. The median is indeed the same for all the measurements concerning the usability of the two versions, except the measurement of the required learning effort to get going with the system (see Table 6.9). In fact, the required learning effort for using OctoUML-V is more than that of OctoUML-Lite (OctoUML-V's median score is 2 against 1.5 of OctoUML-Lite). This is because the subjects had to learn a list of various voice commands that are necessary for using of the voice-interaction-enabled version; OctoUML-V. We have also noticed the learning effort issue during the software design session. Indeed, before starting with the design session, we supplied the subjects with a sheet of paper (short manual) containing a list of the supported voice commands. At the beginning of the session, the subjects often looked to the manual in order to remember the commands. However, after practicing and getting more used to the commands, the subject learned exactly how to master them without consulting the manual.

We found that the employment of voice interaction modality within the software design environments enhances the efficiency of the software design process by reducing the time required to *name* UML class diagram' classes and packages. However, numerous factors (e.g. the distance of the microphone, white noise, human pronunciation, etc.) may affect the effectiveness or accuracy of the voice recognizer, and as a consequence affect the enhancement in the efficiency of the software design process. This is in-line with Mills et al. [95] who pointed out that voice recognition techniques have a high rate of faults and errors which negatively influences their usability. In order to assess this issue, we counted how many voice commands were used during the user study USS2, and how many times the voice recognizer failed to correctly interpret such commands. In particular, we noted:

- *Unintended Commands*. Happen when a given voice command activates/executes a task different from the desired one.

- *Faulty Name Inputs*. Happen when a class or package gets a name different from the one assigned via a voice command of type β (recognition error).
- *Unrecognized Commands*. Stand for voice commands that could not provoke any consequence, in the sense that OctoUML-V could not interpret and even react to such commands.

The average number of used voice commands is 27 (the lowest is 9 and the highest is 42). Overall, the failure rate for using voice commands is 26% (see Figure 4.4). Such a rate was obtained via dividing the total number of voice recognition faults (100) by the total number of executed voice commands (381). Whereas the failure rate for using the voice commands of type β (*faulty name inputs'* failure rate) is 12%. This rate is small, however, it still affects the enhancement in the time required for naming the UML class diagram elements. In order to minimize the failure rate, more sophisticated recognizers are needed to reduce the effects of the factors that may compromise the recognition process.

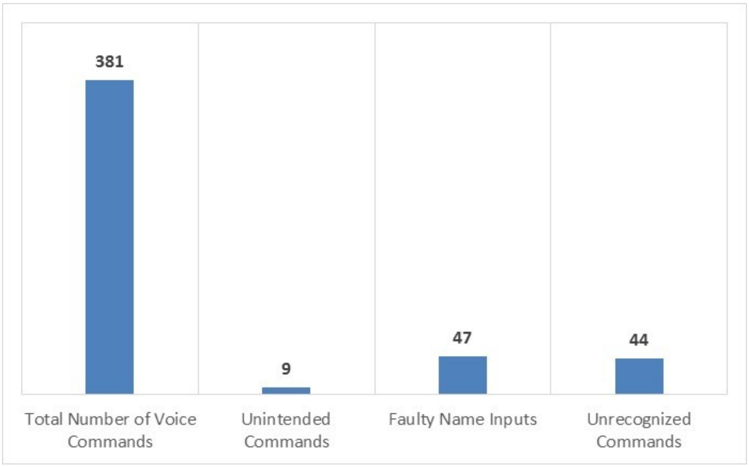


Figure 4.4: Usage and faults of voice commands.

4.7 Threats to Validity

Construct Validity. We assigned a design task which was relatively simple compared to real world problems. This might have influenced the amount of discussions and usability interactions. However during the interviews, all the subjects perceived the potential of OctoUML in managing any kind of software design problems, even complex ones.

Internal Validity. None of the subjects were familiar with OctoUML. To mitigate this, we gave the participants a short introduction explaining the features and functionalities of OctoUML. Moreover, we provided the subjects involved in USS2 with a list of all voice commands that are supported and could be used during the software design session. During the interviews, the participants might have wanted to please the interviewers by giving them a positive feedback. To mitigate this, we asked the participants to answer the SUS questionnaire which allowed them to give feedback anonymously.

External Validity. The involved subjects in two the user studies, USS1 and USS2, may not represent the general population of software engineering community. This could threat the generality of the results. However we involved people with different background, modelling expertise and academical degrees.

4.8 Conclusion and Future Work

Modelling is a common approach for software development as it allows efficient definition of software artefacts in order to create a solution that meets the requirements. There is an evident need for efficient methods and tools for designing software products. Current software design tools constrain the realistic design process rather than supporting it [4]. Furthermore, they lack adaptation and deployment of advanced technologies and need flexibility on both platforms and used input methods.

The main goal of this study was to find out which features of software design environments are prioritized to be supported by voice interaction modality, as well as understand whether the support of such features could: (i) enhance the usability and efficiency of software design environments, and (ii) be of benefit for software design processes. To achieve this goal, we designed and evaluated a multi-modal software design environment, OctoUML, that supports multiple interaction modalities such as touch, mouse, keyboard and voice. Furthermore, we conducted two user studies by involving a population sample of thirty subjects (2 post-docs, 13 PhD and 15 M.Sc. students) to evaluate and compare the usability as well as the efficiency of two versions of OctoUML; one is voice-recognition-enabled (OctoUML-V) and the other is not (OctoUML-Lite).

OctoUML-V was more appreciated by the subjects. Moreover, it also enhanced the efficiency of the software design process by reducing the required time for naming the elements of the software design diagram.

The collected perceptions regarding the usability will be utilized to conduct and track the development progress of OctoUML as more improvements could be done in the future. Furthermore, we will study the impact of employing *multi-touch* and *remote collaboration* techniques in OctoUML, and hence evaluate the usefulness of these techniques in supporting the software design process.

Chapter 5

Paper D

OctoUML: An Environment for Exploratory and Collaborative Software Design

R. Jolak, B. Vesin, M.R.V. Chaudron

In Proceedings of the 39th International Conference on Software Engineering, pp. 7-10. 2017.

Abstract

Software architects seek efficient support for planning and designing models at multiple levels of abstraction and from different perspectives. For this it is desirable that software design tools support both informal and formal representation of design, and also support their combination and the transition between them. Furthermore, software design tools should be able to provide features for collaborative work on the design. OctoUML supports the creation of software models at various levels of formality, collaborative software design, and multi-modal interaction methods. By combining these features, OctoUML is a prototype of a new generation software design environment that aims to better supports software architects in their actual software design and modelling processes.
Demo video: https://youtu.be/fsN3rfEAYHw
OctoUML Project: https://github.com/Imarcus/OctoUML

Keywords: software design; modelling notations; multi-modal interaction; collaborative design; user experience; UML

5.1 Introduction

Designing software consists of exploring design problems, discussing solutions and creating software models as design artifacts. Such artifacts provide a bridge between problem and software implementation by describing user's needs as well as the product to be developed. As software systems are gaining increased complexity, the importance of efficient software design tools is also increasing. Software models change frequently and are quite often updated by many designers simultaneously [66]. These models should present a description of complex systems at multiple levels of abstraction and from a different perspectives. Therefore, it is crucial to provide software design tools that give possibilities for efficient and collaborative development as well as options for multi-modal interaction.

Modelling tools can be classified into two groups: informal and formal [71]. We mean by informal any tool that supports informal design in the sense that it does not constrain the notation used. Indeed, informal tools are preferred for their flexibility as well as the role that they play in unleashing designers' expressiveness. Examples of such tools are whiteboards, paper and pencil. While we mean by formal any tool that support one or few formalized notations. Typical examples are UML CASE-tools (e.g. Rational Rose, Enterprise Architect, Papyrus, StarUML, etc.). Formal tools are usually used for code-generation and/or documenting purposes.

During early design phases, software designers often use informal tools (e.g. whiteboards) to sketch their thoughts and compare design ideas. Once the designers settle on one possible solution, they proceed to create a formal version of the sketchy design. In particular, they move from the whiteboard, start-up the computers, run a formal tool (a CASE-tool), and re-enter the solution that has been created previously during the early design phase. So there is a gap between informal designing in early software design phases and formal design and documentation practices in subsequent development. To bridge this gap, we present *OctoUML*, a software design environment that supports exploratory and collaborative design meetings. OctoUML provides means to allow the creation of both sketchy hand-drawn elements and formal notations simultaneously. Moreover, it allows the transformation of sketchy designs into formal notations.

Oviatt and Cohen [101] illustrated the importance of multi-modal systems in reshaping daily computing tasks and predicted their future role in shifting the balance of human-computer interaction much closer to the human. We enabled OctoUML to support multiple modes of interaction including mouse, keyboard, touch/multi-touch using fingers and styluses, sketching, and voice modality.

More often than not, the process of software design involves several designers working on the same project simultaneously. This could also occur in usercentered design situations where users are involved in the design process. We implemented OctoUML to support design collaborative sessions, both *in-situ* (via the adoption of multi-touch technique) and at a distance *"remotely"* (by using a client-server paradigm). OctoUML can be run using a number of input devices ranging from desktop computers over large touch screens to large interactive whiteboards.

The paper is organised as follows: the related work is presented in section two. Further information on OctoUML, its architecture and features, and the performed evaluation are reported in section three. The future objectives and concluding remarks are presented in the last section (section four).

5.2 Related Work

Several studies proposed different approaches to enhance the software design process. Mangano et al. [51] identified some behaviors that occur during informal design. In particular, designers sketch different kind of diagrams (e.g. box and arrow diagrams, UI mock-ups, generic plots, flowcharts, etc.) and use impromptu notations. The authors implemented an interactive whiteboard system (called *Calico*) to support these behaviors and identified some ways where interactive whiteboards can enable designers to work more effectively.

Wüest et al. [74] stated that software engineers often use paper and pencil to sketch ideas when gathering requirements from stakeholders, but such sketches on paper often need to be modelled again for further processing. A tool, *FlexiSketch*, was prototyped by them to combine free-form sketching with the ability to annotate the sketches interactively for an incremental transformation into semi-formal models. The users of *FlexiSketch* were able to draw UML-like diagrams and introduced their own notation. They were also able to assign types to drawn symbols. Users liked the informality provided by the tool, and had the will to adopt it in practice.

Magin and Kopf [56] created a multi-touch based system allowing users to collaboratively design UML class diagrams on touch-screens. They have also implemented a new algorithm to recognize the gestures drawn by the users and to improve the layout of the diagrams. However, their tool does not allow for informal freehand sketching of arbitrary notations.

Lahtinen and Peltonen [81] presented an approach to build speech interfaces to UML tools. The authors set up a spoken language to manipulate the UML models, and built a speech control system (*VoCoTo*) integrated with a CASE-tool (*Rational Rose*). They stated that speech recognition is applicable to be used to enhance the interaction with UML tools.

Table 5.1 summarizes the main supported functionalities by OctoUML and illustrates the differences to the related work.

Table 5.1: Comparison between OctoUML and the related work.

Related Work	**Informal & formal notations**	**Interaction Modalities**	**(Multi-Touch, Remote Control)**
Calico	informal hand-drawn notations	mouse, keyboard and touch	(no, no)
Flexisketch	informal hand-drawn notations	mouse, keyboard and touch	(no, no)
Magin&Kopf	formal notations creation via gestures	touch-based	(yes, no)
VoCoTo	formal notations	mouse, keyboard and voice	(no, no)
OctoUML	creation and mix of informal and formal notations simultaneously	mouse, keyboard, single touch, multi-touch, and voice	(yes, yes)

5.3 OctoUML

In a previous work [71], we presented our vision for a new generation software design environment. To realize our vision, we developed a prototype called OctoUML [102]. OctoUML is a software design environment that supports exploratory and collaborative software design. It is used to create and organize diagrams as well as supports their modification and evolution. Firstly, we illustrate the architecture of OctoUML. Secondly, we describe the main functionalities that are supported by OctoUML (sections B and C). Later on, we provide a scenario showing how such functionalities could support the design process. Lastly, we provide some details on OctoUML evaluation.

5.3.1 OctoUML's Architecture

The key architectural components of OctoUML are presented in Figure 5.1. The environment contains three major components: *UI component*, *Data component* and *Services*. The current version of the system offers only the *UI* and *Data* components. Additional services will be added during future development. The *UI component* consists of: *Presentation manager* and *Input unit*. The *Presentation manager* provides means for performing stylus or touch-based input commands on devices being used. *Drawing layers* include support for both informal and formal modelling layers. The *Command tools* are responsible for transferring the inputs from users to different controllers. The *Graph controller* allows switching between different input techniques with combining of multiple layers. The *Input unit* is responsible for processing different inputs. In particular, a *Sketch recognizer* is provided to recognize and transform informal models into formal concepts, and hence allows to

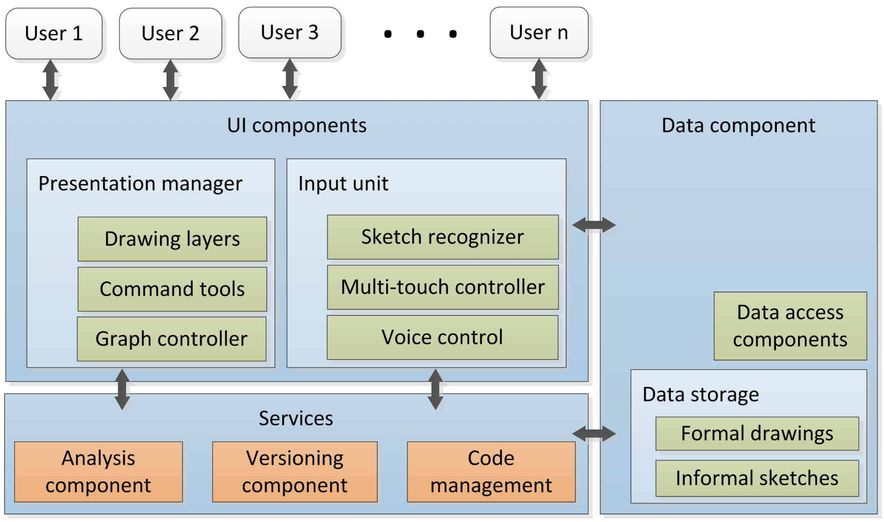


Figure 5.1: Architectural Components of OctoUML

maintain and transfer the designs for further processing tasks. A *Multi-touch* controller captures and coordinates the inputs from different touch-points. All the program data are saved and stored in the *Data component*. Our tool uses a set of data structures to manage and maintain the sketched elements and formalized designs.

5.3.2 Informal and Formal Notation

Whiteboards (or any informal tools e.g. paper and pen) are used during early software design phases because of their flexibility and immediacy, but also becuase they do not constrain the notation being used. Informal notations (e.g. *sketches*) can be used to express abstract ideas representationally, to allow checking the entirety and the internal consistency of an idea as well as to facilitate development of new ideas [69]. Furthermore, informal notations can have a very close mapping to the problem domain. However, the informal notations often need to be formalized in order to allow their manipulation and process e.g. sharing, code generation or documentation.

Modelling tools should not constrain designers to create only some specific notations. Furthermore, they should maintain the characteristics of formal tools in their support of design transfer and persistence [71].

OctoUML allows the creation of both hand-drawn informal sketches and computer-drawn formal elements (currently UML class and sequence models)

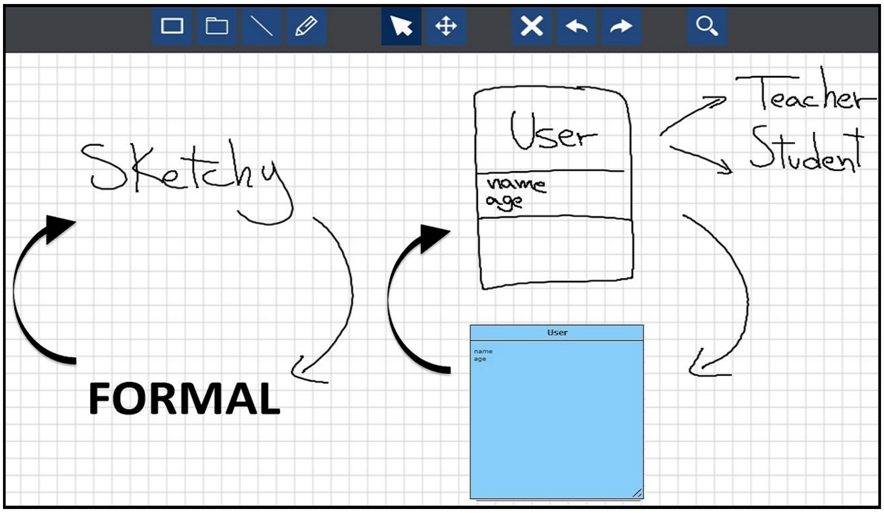


Figure 5.2: Combination of different notations on the same canvas

on the same canvas simultaneously (Figure 5.2). OctoUML bridges the gap between early software design process, when *informal* tools are typically used, and later documentation and formalization process, when *formal* tools are used. Beside supporting the creation of software models at different levels of formality, OctoUML is equipped with a *Sketch recognition unit* which enables sketch formalization. In particular, OctoUML allows the transformation of models from informal to formal and vice versa at any time during the modelling session. Furthermore, we adopted a layering technique by which the informal notations belong to one layer that we call the *informal layer*, and the formal notations belong to another layer that we call the *formal layer*. The user can then select to see the layers in combination or isolation.

5.3.3 Interaction Modes and Collaboration

The usability of current CASE tools is a common source of criticism [7]. The interaction with such tools is often based on using the mouse and keyboard. Other modes of interaction (e.g. touch, gesture and voice) could be more natural and intuitive. In order to improve the user experience of OctoUML and increase its accessibility, the interaction modalities of OctoUML are enriched by providing a *voice-commands* recognition component capable of transforming designers' voice-commands into control actions.

The process of software design often involves more than one designer working on the same project simultaneously. OctoUML promotes collaborative design by adopting a *multi-touch* technique and supporting *remote collaboration.* Next, we provide more details on the supported functionalities:

- *Multi-touch* is an interaction technique that permits the manipulation of graphical entities by more users at the same time. Our tool allows multiple users to design diagrams simultaneously by performing simple touch gestures.
- In order to improve the user experience, we integrated a *voice-commands* control component within the *Input unit.* The component is capable of handling the most commonly used functions during the design process. Thus, users can use voice commands in order to create and edit elements of software diagrams.
- To open up new opportunities for interactive collaborative design, our tool supports *remote collaborative* sessions between geographically distributed teams. One team of designers can run a server instance of OctoUML, whereas another team can join the session as client connecting to the server. Video calls and chatting tools will be integrated in order to support the joint design sessions.

5.3.4 Design process in UctoUML: A Scenario

Figure 5.3 illustrate the design process in OctoUML. Activities that are currently supported by OctoUML are distinct in green. Let us think about the following scenario: a group of software designers meet to explore and discuss design ideas of a specific software product. The designers start with the creation of some informal sketchy designs using OctoUML being deployed on a large interactive whiteboard. After that, the designers proceed with a selective transformation of some informal sketches into a formal model. Later on, the created model is analyzed to check possible flaws and performance bottlenecks. Finally, the model is saved and uploaded to a version control repository. The designers meet again (on-site or from different locations) when new requirements come out or having earlier requirements exposed to changes. They fetch the design that was previously shared on the version repository, update the design, and commit a new version that is now compliant to the new requirements.

5.3.5 Evaluation

Two *user studies* were performed to evaluate OctoUML. In both studies, the participants had to do a modelling task using OctoUML, answer a System Usability Scale (SUS) questionnaire [77], and participate into semi-structured interviews. The first study involved fourteen software engineering students (ten PhD and four M.Sc. students) and two post-doc researchers. The main

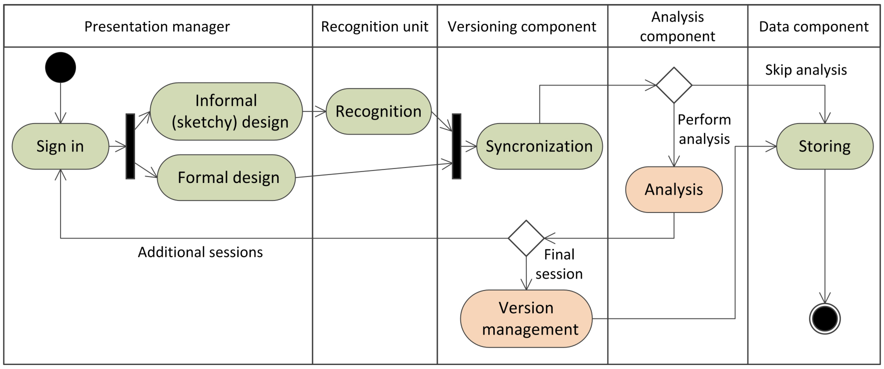


Figure 5.3: Design process in OctoUML

purpose of the first study was to evaluate the usability of OctoUML as well as to investigate whether supporting the mix of informal and formal notation could support the design process. OctoUML got an average SUS-score of 78.75 which is a high usability score according to [43]. The participants stated that informal notations could be valuable artifacts beyond being just explorative means. They also stated that such notations support designers' activities in understanding the problems and communicating ideas. Figure 5.4 shows the feedback from the participants regarding the use of informal and formal notations within OctoUML.

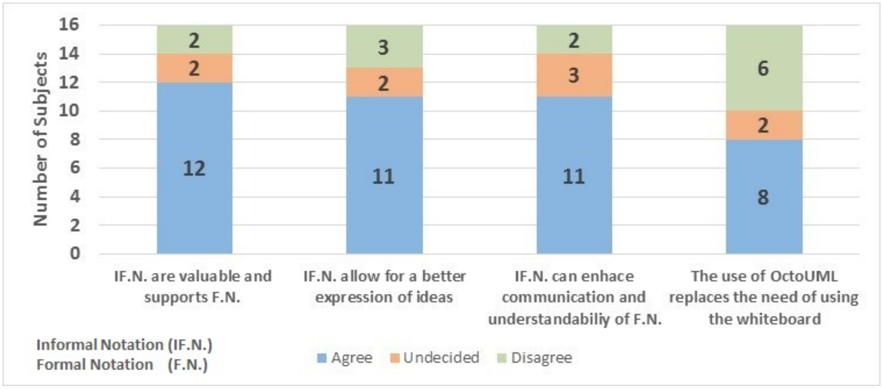


Figure 5.4: User study I: informal vs. formal notations

The second study involved fourteen participants (three PhD and eleven M.Sc. software engineering students). The main purpose was to evaluate the learnability and usability of OctoUML as well as the role of the voice interaction

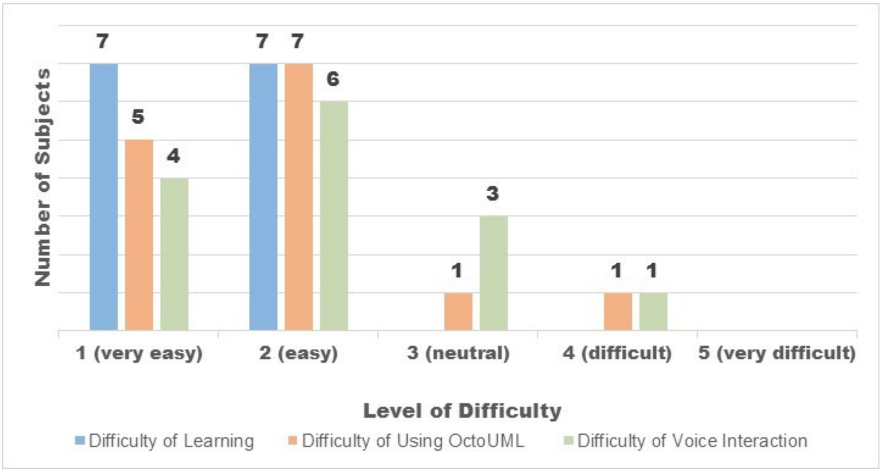


Figure 5.5: User study II: usability and learnability of OctoUML

modality in enhancing the user experience and supporting the software design process. OctoUML got a SUS-score of 74.6 which can be considered a quite good usability score [43]. The majority of the participants stated that it was easy to learn and use the different functionalities of OctoUML (including the voice interaction modality), see Figure 5.5. Furthermore, the voice interaction modality was perceived helpful in overcoming non-ergonomic tasks e.g. typing via a keyboard.

5.4 Conclusion and Future Development

In this paper we presented OctoUML, a prototype of a new generation software design environment for collaborative software design. It provides support for mixing informal hand-drawn elements with formal notations. Moreover, it supports different input methods and interaction modalities.

OctoUML combines the advantages of both informal tools e.g. interactive whiteboards and formal tools e.g. CASE tools, and therefore is able to bridge the gap between early software design process (when designers often sketch their ideas) and formalisation/documentation process. OctoUML was evaluated by conducting two user studies and involving thirty participants in total. The main goal was to get feedback on the viability and usability of OctoUML. The results show that the participants enjoyed their experience with OctoUML and had a positive perception regarding its usability.

The current architecture of OctoUML allows future expansions of the system with additional functionalities. The goal is to implement and incorporate additional features in the subsequent versions of the system:

- *Analysis component.* It will perform software model analysis. This tool will be used to automatically evaluate the created software models to detect general design flaws, security flaws and performance bottlenecks.
- *Versioning component.* The purpose is to provide a repository for keeping track of the version history of stored models, and the ability to observe changes that are made to specific artifacts in the environment. The system should also be able to resolve conflicts when two users change the same model data. Such component would increase the potential for parallel and distributed work, improve the ability to track and merge changes over time, and automate management of revision history. It would also allow multiple designers to work concurrently, supporting tight collaboration and a fast feedback loop.
- *Code management.* Models and code must be combined throughout the development process. Users will be able to generate code from formalized UML class diagrams as well as view models and codes side by side and jump between editing one and keeping the other synchronized.

Chapter 6

Paper E

Dissecting Design Effort and Drawing Effort in UML Modeling

R. Jolak, E. Umuhoza, T. Ho-Quang, M.R.V. Chaudron, M. Brambilla

In Proceedings of 43th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. in print. 2017.

Abstract

One argument in the discussion about the adoption of UML in industry is the supposedly large effort it takes to do modeling. Our study explores how the creation of UML models can be understood to consist of different cognitive activities: (i) *designing*: thinking about the design (ideation, key-design decision making), (ii) *notation expression*: expressing a design in a modeling notation and (iii) *layouting*: the spatial organization of model elements in a diagram. We explain that these different subactivities relate to different short-term and long-term benefits of modeling. In this study we present two controlled experiments with a total of 100 subjects creating models for a small system. In these experiments we focus on software models as represented through UML class diagram. Our results show that *at least* 56% of the effort spent on creating a class model is actually due to *designing*. *Notation expression* is around 41% of the model creation effort and *layouting* is in the order of 3%. This finding suggests that a significant part of creating models is devoted to design thinking about the problem.

Keywords: Software Engineering; Software Modeling; Software Design; Modeling Effort; Designing Effort; Design Thinking; UML

6.1 Introduction

Models have emerged in software engineering as a powerful tool to tackle complexity of system specifications. Indeed, modeling allows to address the description of software based on different levels of abstraction and from multiple perspectives, in order to accommodate the needs of communication and description of a variety of stakeholders. In fact, models help to describe, reason, predict and evaluate both software problems and solutions. Furthermore, they provide effective means for supporting the communication between stakeholders, and serve as specifications for implementation [103]. However, software practitioners consider such approaches *time-consuming*, hence prefer to avoid using models which are believed as complex, inconsistent, excessive and unnecessary artifacts [18].

In this study we want to find out how much of the modeling effort is spent on the *design* of the solution (i.e. pondering and making the design decisions). If a significant part of the creation of the models is devoted to design thinking about the problem, it could means that the fault of supposedly unproductive processes should not be blamed on modeling, but to the (anyhow necessary) effort devoted to thinking about the problem and identifying the solution (i.e. *design* effort).

In order to assess this, we run a set of experiments about the creation of models in response to a set of requirements: we measure the effort required to make the initial model of a system (*modeling* effort), and then we measure the effort required to recreate the same model again, simply by redrawing the already defined solution (*copying* effort). For the *copying* of the solution, we assume a subject does not have to (re)do any design thinking, but only spend effort on entering a solution into a modeling tool. At the end we calculate *design* effort by assessing the time difference between the two activities (see the details in Section 6.3).

Some empirical studies provided evidence on the benefits of UML modeling in enhancing the productivity, quality and maintenance of software products [15, 80, 104, 105]. These studies sustain that benefits of UML modeling take place after a *long-term*, in the sense that UML modeling introduces an initial overhead at the beginning, whereas the benefits start to take place at late stages – like the *two* marshmallows reward of the *'one marshmallow now or two marshmallows later'* experiment [106]. The minority of the kids who participated in that experiment preferred to have one marshmallow now rather than two later. Such behaviour may be similar to that of software developers' who do not prefer to spend time on modeling at early stages, and eat the marshmallow immediately.

Our hypothesis is that the benefits of UML modeling does not only take place after a long-term, but also immediately at early stages. To assess this hypothesis, we try to dissect *design*, *notation expression* and *layout* efforts in UML modeling. If *design* effort dominates the modeling process, then UML modeling consists mostly of thinking of the domain problem and identifying/designing the solution. In other words UML modeling would reward *three* marshmallows; one immediately (foster design thinking and promote ideation) and two later (enhance productivity, quality and maintenance).

This paper is organized as follows: in Section 6.2 we discuss the related work. We describe our approach in Section 6.3. Section 6.4 illustrates the design of our experiments and details their operational phases. Section 6.5 reports the results of the experiments, which are then discussed in Section 6.6. We consider the threats to validity in Section 6.7. Finally, we conclude and discuss the future work in Section 6.8.

6.2 Related Work

More often than not, evaluation of software modeling practices and associated effort have been left in the realm of myth. As a result, software developers and also modeling experts have different opinions on the pros and cons of modeling that rely on beliefs (i.e. factoids) more than facts. This leads to a variety of situations where no proper guidance can be provided in the selection of the appropriate design tools for software. More in general, the whole field of software engineering perceives the discrepancy between scientifically validated results (e.g., in the empirical software engineering) and developers beliefs, usually based only on personal perspectives on the development processes. Various recent studies demonstrate that more in-depth studies that address the interplay of belief and evidence in software practices are needed [107].

A few studies addressed the monitoring and analysis of modeling practices. Sharif et al. [33] explored design strategies and types of activities that designers engage in during software design sessions. They used video-recordings and transcripts of three two-person teams who were assigned to create a software design for the same set of requirements on a whiteboard. In addition to the identified design activities, they also found the sequence of activities for each session as well as the activity that took the longest by analyzing the duration of actions and speeches mapped to various design activities. They identified some of time-consuming activities such as decisions about the logic, discussion of uses cases, drawing class diagrams, and drawing the user interface.

Some experiments have been conducted to identify strategies during the modeling task. The works [108,109] recorded the activities of a pool of students when creating UML class diagrams and analysed the logs using LogViz. Four different strategies were found, namely Depth First, Breath First, Depthless and Adhoc. The study also found that students spent most of their time in understanding assignment tasks and in defining the layout of the model. One thing that the log failed to say was what students do in the time gaps where they appeared to do “no activity”.

Despite the resistance of software companies in adopting the model-driven development, few industrial success stories can be found. Brambilla and Fraternali [110] presented the industrial success stories (spanning from financial and banking to utility) and the advantages of adopting Model-Driven Engineering (MDE) perceived by the customers of WebRatio, a company which focuses on MDE tools and services since 2001. The study also included a report of amount of the effort dedicated by the designers to the different modeling activities.

Furthermore, few researches evaluated some of the claimed advantages brought by MDE. Diaz et al. [111] measured the reuse gains brought by MDE in comparison with manual coding of blogs. Brambilla et al. [112] analyzed the productivity gain brought by MDE in comparison of manual coding of cross-platform mobile applications. They observed that for mobile applications, model-driven development allows to save more than 20% of the cost.

6.3 Approach

In this section, we provide an overview of the approach used to estimate the effort devoted to different activities in the process of software modeling. Our approach considers software modeling as a process that encompasses three different activities:

[a] Designing of the solution: It represents the activity of reasoning and thinking about a design solution of a domain problem. We call the time devoted to this activity: *Design Effort (DE).*

[b] Notation expression: The expression of the identified solution through a modeling notation. We call the time devoted to this activity: *Notation Expression Effort (NEE).*

[c] Layouting: It represents the activity of organization of the model elements in a diagram (e.g. to enhance the readability of the model). We call the time devoted to this activity: *Layout Effort (LE).*

Based on that, the total effort dedicated to the software modeling process is simply obtained as a sum of the single efforts spent in each modeling activity. The total Modeling Effort *(ME)* in given by Equation Eq.A.

$$ME = DE + NEE + LE \tag{Eq.A}$$

To compute the effort spent in each activity, we ran two-phase experiments. In the first phase, we measure the effort required to make the initial model of a system (*modeling* effort). While in the second phase, we measure the effort required to recreate the same model again, simply by redrawing the already defined solution (*copying* effort). At the end, we calculate the *design*, *notation expression*, *layout* efforts by assessing the time difference between the two phases.

6.3.1 Phase 1: Modeling

During this phase the participants are asked to create a UML class diagram that addresses a simple assignment using a modeling tool. The participants think about the solution, express their solution through a modeling notation, and may organize the elements of the model on the canvas.

Let us denominate the set of all persistent elements that are part of the final model with (Π), and the set of all deleted elements that are not in the final model with (Δ). Let us also denominate the set of all elements (persistent and deleted) with ($\varSigma$). We have that:

$$\varSigma = \Pi \cup \Delta \tag{Eq.B}$$

Based on Equation Eq.A and Equation Eq.B, the effort dedicated to the modeling phase, called *modeling effort (ME)*, is given by the following equation *(unknowns are in bold, whereas the known efforts are obtained via analyzing the log of the modeling tool)*:

$$\mathrm{ME}(\varSigma) = \mathbf{DE}(\varSigma) + \mathbf{NEE}_m(\Pi) + \mathbf{NEE}_m(\Delta) + LE_m(\Pi) + LE_m(\Delta) \tag{Eq.1}$$

Where:

- $\mathrm{ME}(\varSigma)$: *(known)* the total modeling effort,
- $\mathrm{DE}(\varSigma)$: *(unknown)* design effort, the time spent on thinking about the solution (including both persisted and deleted elements),
- $\mathrm{NEE}_m(\Pi)$: *(unknown)* notation expression effort of persistent elements during modeling phase; the time spent on creating elements that are part of the final model,
- $\mathrm{NEE}_m(\Delta)$: *(unknown)* notation expression of deleted elements during modeling phase; the time spent on creating elements that are not in the final model (deleted because of exploring design alternatives),
- $\mathrm{LE}_m(\Pi)$: *(known)* layout effort of persistent elements during modeling phase; the time spent on organizing elements of the final model,
- $\mathrm{LE}_m(\Delta)$: *(known)* layout effort of deleted elements during modeling phase; the time spent on organizing elements that are not in the final model.

6.3.2 Phase 2: Copying

During the copying phase, the participants are asked to simply re-draw (copy) the same modeling solution produced in phase 1. In this phase the participants are asked to do a strict copy without thinking or enhancing the identified solution in phase 1. Thus, the effort dedicated to this phase, called *copying effort (CE)*, is obtained via the following equation:

$$CE(\Pi) = \mathbf{NEE}_c(\Pi) + LE_c(\Pi) \tag{Eq.2}$$

Where:

- $\mathrm{CE}_c(\Pi)$: *(known)* the total copying effort,
- $\mathrm{NEE}_c(\Pi)$: *(unknown)* notation expression effort of persistent elements during copying phase,
- $\mathrm{LE}_c(\Pi)$: *(known)* layout effort of persistent elements during copying phase.

6.3.3 Analyze Effort Difference

By isolating *design* and *layout* efforts, the *notation expression* effort of persistent elements is the same in both phase 1 and 2. We have that $\mathrm{NEE}_m(\Pi) = \mathrm{NEE}_c(\Pi)$. We compute the *design* effort by subtracting Equation Eq.2 from Equation Eq.1. The final result is reported as follows:

$$\mathbf{DE}(\Sigma) = ME(\Sigma) - CE(\Pi) - LE_m(\Pi) - LE_m(\Delta) - \mathbf{NEE}_m(\Delta) + LE_c(\Pi) \tag{Eq.3}$$

First of all, $\mathrm{LE}_m(\Pi) + \mathrm{LE}_m(\Delta) = \mathrm{LE}(\Sigma)$ is the total layout effort (LE) in the modeling phase. Now let us consider the number of persistent and deleted elements in the modeling phase as $|\Pi|$ and $|\Delta|$, respectively. Based on Equation Eq.2, we identify $\mathrm{NEE}_m(\Delta)$ via the following equation:

$$\mathbf{NEE}_m(\Delta) = \frac{|\Delta|}{|\Pi|} \cdot \mathrm{NEE}_c(\Pi) \tag{Eq.4}$$

The (DE) is given by inserting the value of $\mathrm{NEE}_m(\Delta)$ in Equation Eq.3. Furthermore, considering Equation Eq.A, the notation expression effort is given by the following equation:

$$NEE = ME - DE - LE \tag{Eq.5}$$

We are interested in identifying how much of the total modeling effort is spent on *designing*, *notation expression* and *layouting*. Thus, we can define the

Design Effort Percentage (DEP) as the ratio of the Design Effort (DE) over the total Modeling Effort (ME):

$$DEP = DE/ME \qquad \text{(Eq.6)}$$

Similarly for NEE and LE, the percentages are given by:

$$NEEP = NEE/ME \qquad \text{(Eq.7)}$$

$$LEP = LE/ME \qquad \text{(Eq.8)}$$

At this point, we want to underline that the DE may also occur during the process of UML notation expression and/or layouting (as we are capable of thinking while drawing). Our calculations indeed estimates the lower bound on DEP (the *minimum* DEP), in the sense that we do not assume any occurrence of DE during the process of notation expression and/or layouting. So the real value of DEP may be more than the minimum found. At maximum, DE could occur continuously from the beginning to the end of the modeling process (i.e. Max(DEP) is 100%).

Our experiments were conducted at Polytechnic University of Milan in Italy and Gadjah Mada University in Indonesia. We formulated three different modeling scenarios. Every scenario describes a system to be designed (see Section 6.4.1.1). A mix of 100 B.Sc. and M.Sc. software engineering students were randomly given the modeling scenarios. To create their models, students were asked to use WebUML [108] and Papyrus (https://eclipse.org/papyrus) modeling tools. Both tools allow the logging of the modeling activities. We collected the recorded log files for each participant and assignment. We also setup an online questionnaire through which participants answered questions about their background, expertise in UML modeling, tool usability, and assignments understandability.

Based on the collected results, our research objective has been addressed by defining and responding to the following research questions:

[a] How much of the modeling effort is *design*, DEP?

[b] How much of the modeling effort is *notation expression*, NEEP?,

[c] How much of the modeling effort is *layout*, LEP?

[d] Does the *size* of the modeling scenario affect DEP, NEEP and LEP?

[e] Does the *topic* of the modeling scenario affect DEP, NEEP and LEP?

6.4 Experiment

This section describes the modeling experiments conducted to answer the research questions presented in Section 6.3. For this study, we conducted the experiments in two different settings: (i) participants create models *Individually* and (ii) in teams *Collaboratively*. We refer to the former setting as *EXP1* while the latter as *EXP2*.

EXP1 was conducted at Polytechnic University of Milan involving 48 students. During this experiment, the participants were asked to design a solution for given modeling scenarios using the *WebUML editor*. EXP2 was conducted in Gadjah Mada university in Indonesia involving 13 groups of 4 students each. Each group was asked to design a solution for a given modeling scenario using Papyrus.

6.4.1 Experiment Preparation

6.4.1.1 Scenarios Definition

To make our analysis of the specific case independent, we evaluated the *design* effort with three scenarios (*scenario 1*, *scenario 2* and *scenario 3*) from different topics and with slightly different size (the number of classes in their solution is different by one or two). Every scenario describes a simple system to be designed. In addition, we have defined a *test scenario*, used at the beginning of the modeling sessions to explain to the participants how the tool works and to let them get familiar with it. The description of the scenarios and the experimental material can be found here: (`https://goo.gl/mvz2bm`).

6.4.1.2 Assigning scenarios to participants

EXP1 The ideal strategy to get the most generalizable results is to assign all the three scenarios to each participant. However considering the average time required to complete one scenario (around 25 minutes), assigning three scenarios to each participant was not feasible due to the limited time the students had available for the experiment. Thus, we decided to assign two scenarios to each participant. In order to limit unintended effects and to have balanced experiments, we used the Graeco-Latin square theory [113] by assigning different orders of scenario's to different groups of students. The scenarios that are used in this experiment are: scenario 1, scenario 2 and scenario 3.

EXP2 The purpose of this setting is to study possible effects of group work and modeling tools on the software modeling effort. The used scenario in this experiment is: scenario 2. The obtained data from this experiment are compared to the data that are related to scenario 2 of EXP1.

6.4.2 Experiment Execution

In this phase, the participants model the assigned scenarios using the WebUML tool for *EXP1* and Papyrus for *EXP2*. Both tools have a logging feature that logs the participants' actions (such as the creation, modification, and deletion of an element). The logs are useful to derive quantitative data which enable us to compare and evaluate the produced designs and the time spent on interacting with the tool. The experiments were conducted following these five steps:

[a] *Introduction.* We introduced the modeling tool to the participants through a training session.

[b] *Instruction.* During this phase we explained the procedure of the experiment to the participants, as well as showed them how to save and submit their designs. Then, a short training exercize of 15 minutes took place under our supervision in order to get the participants accustomed with the basic functionality of the tool. This was done using a test scenario, equal for all the participants.

[c] *Modeling assigned scenarios* (See Section 6.3, *Modeling phase*). The participants have to model the assigned scenario/s.

[d] *Copying assigned scenarios* (See Section 6.3, *Copying phase*). The participants simply copy (re-draw) the proposed model solution that is produced in the *Modeling Phase.*

At the end of step 4, the participants of *EXP1* were asked to proceed with the *modeling* and *copying* of the other scenarios following the same way as described in steps 3 and 4, respectively.

[e] *Closure.* after submitting their models, the participants were asked to answer a questionnaire about their personal information, knowledge about UML modeling and perception regarding the usability of the modeling tools (WebUML for *EXP1* and Papyrus for *EXP2*).

The experiments were performed in a controlled environment. The participants worked on computers in a lab at both Universities. There were supervisors that walked around to monitor that the participants worked on the assignment and not on other tasks or distractions.

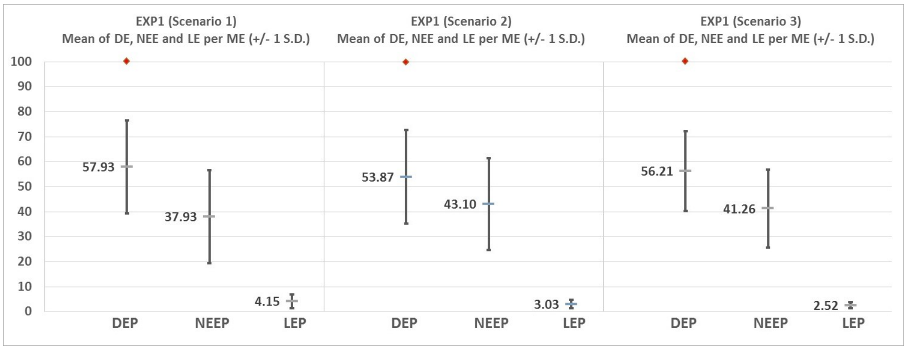


Figure 6.1: Efforts distribution related to the scenarios of EXP1

6.5 Results

In this section we report the results of the two conducted experiments (EXP1 and EXP2). For EXP1, we present the results of 37 subjects since the experiments of 11 subjects were not valid (8 worked concurrently on the first and second task while 3 had technical network problems which prevented us from receiving their data) and then removed from the data set. We used the statistical package R [114] to perform all tests. We chose a significance level at 0.05, which corresponds to a 95% confidence level.

6.5.1 Design, Notation Expression and Layout Efforts

6.5.1.1 EXP1

For each modeling scenario used during EXP1, we calculated the mean and the standard deviation of DEP, NEEP and LEP. The results are presented in Table 6.1.

6.5.1.2 EXP2

We calculated the mean and the standard deviation of DEP, NEEP and LP that are related to scenario 2 used in experiment EXP2. The results are presented in Table 6.1. Figure 6.1 and 6.2 provide a better view of the distributions of the various efforts (DEP, NEEP and LEP) related to the scenarios used in EXP1 and EXP2, respectively. (Note that the *red diamond* represents the Max(DEP), as the DE may occur concurrently with notation expression and/or layouting activities.)

Table 6.1: Statistical results for all scenarios of EXP1 and EXP2

EXP1							
Scenario	**# Subjects**	**DEP**		**NEEP**		**LP**	
		Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Scenario 1	21	57.93	18.66	37.93	18.65	4.15	2.69
Scenario 2	25	53.87	18.72	43.10	18.39	3.03	1.75
Scenario 3	22	56.21	16.03	41.26	15.67	2.52	1.20
All		55.88	17.69	40.91	17.51	3.21	2.04
EXP2							
Scenario	**# Subjects**	**DEP**		**NEEP**		**LP**	
		Mean	Std. Dev.	Mean	Std. Dev.	Mean	Std. Dev.
Scenario 2	52	73.60	8.40	22.14	8.34	4.25	2.88

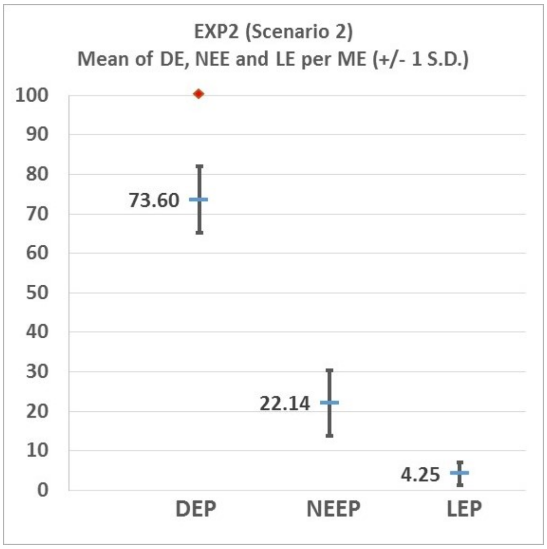


Figure 6.2: Efforts distribution related to scenario 2 of EXP2

Results for Q1, Q2 and Q3 (EXP1): For all scenarios, we found that the average *Design Effort* is 55.88%. While the rest is: 40.91% *Notation Expression Effort* and 3.21% *Layout Effort.*

Results for Q1, Q2 and Q3 (EXP2): the average ***Design Effort*** for **scenario 2 is 73.60%. While the rest is: 22.14% *Notation Expression Effort* and 4.25% *Layout Effort.***

Table 6.2: Quality of the models

						I-R. R.	
EXP	**Scenario**	**M**	**Q1**	**Q3**	**I-Q. R.**	***Kappa***	**p-value**
1	Scenario 1	4.00	3.00	4.00	1.00	0.57	0.000
1	Scenario 2	3.00	3.00	4.00	1.00	0.39	0.000
1	Scenario 3	4.00	3.00	4.00	1.00	0.61	0.000
2	Scenario 2	4.00	3.00	4.00	1.00	0.53	0.002

6.5.1.3 Quality of the models

we consider the quality of the produced models as a further crucial factor. We wanted to know how well the models reflect the domain problem, because extremely bad (or rough) models could affect the effort statistics. To grade the quality of the models, we defined a rubric before running the experiment. The rubric consists of a 5-point scale grading guidelines (`https://goo.gl/mvz2bm`). In advance of the actual grading a set of possible ideal solutions was discussed. The grading was done in two steps: (i) the assessors graded all models separately (independently), (ii) the assessors discussed the differences in grading and gave the model the final mark. Cohen's κ [115] was run to determine if there was agreement between the assessors. See Table 6.2 (the range of ratings is [1 to 5] where 1 is the most negative score and 5 is the most positive score, ratings are reported in terms of: (EXP: Experiment, M: Median Score, Q1: 1st quartile, Q3: 3rd quartile, I-Q. R.: Inter-Quartile Range (Q3-Q1), I-R. R.: Inter-Rater Reliability).

6.5.2 Comparison between the results of EXP1 and EXP2

The same modeling scenario (*scenario 2*) was used in both experiments EXP1 and EXP2. However, there are two factors that could affect the DEP, NEEP and LEP: (α) the modeling tool and (β) number of involved subjects per modeling task. For EXP1, *WebUML* was used by individuals. Whereas for EXP2, *Papyrus* was used by thirteen groups (of 4 people each). In order to assess the effect of both factors (α and β) on the DEP, NEEP and LEP, we performed Mann-Whitney's non-parametric test as the data are not normally distributed (Shapiro-Wilk test' p-values are less than 0.05). The following hypotheses were formed:

- Null Hypothesis H_{01}: There is no statistically significant difference in the DEP of the two cases.
- Alternative Hypothesis H_{A1}: There is a statistically significant difference in the DEP of the two cases.

Table 6.3: Impact of modelling tool and collaboration on DEP, NEEP and LEP (Mann-Whitney tset)

Data	**Mann-Whitney U**	**sig. 2-tailed**
DEP	52.000	0.001
NEEP	42.000	0.000
LEP	125.000	0.249

Table 6.4: Impact of the size of models (Kurskal-Wallis test)

Data	**Chi-square**	**df**	**p-value**
N of Classes	4.151	2	0.126
N of Associations	4.151	2	0.126

- Null Hypothesis H_{02}: There is no statistically significant difference in the NEEP of the two cases.
- Alternative Hypothesis H_{A2}: There is a statistically significant difference in the NEEP of the two cases.
- Null Hypothesis H_{03}: There is no statistically significant difference in the LEP of the two cases.
- Alternative Hypothesis H_{A3}: There is a statistically significant difference in the LEP of the two cases.

Table 6.3 shows the results of the test. Since the p-values of DEP and NEEP are less than 0.05, we reject the null hypotheses (H_{01} and H_{02}) and accept the alternative hypotheses (H_{A1} and H_{A2}). In other words, the differences between the mean rankings of DEP and NEEP of the two cases are statistically significant. The p-value of LEP is $0.249 > 0.05$. We cannot reject the null hypothesis (H_{03}), and the difference between the mean rankings of LEP of the two cases is not significant.

We have a statistical evidence to conclude that the DEP and NEEP are affected by the change of both the modeling tool and the number of involved subjects per task. While LEP is not affected by such change.

6.5.3 Impacts of The Topic/Size of The Modeling Scenarios on DEP, NEEP and LEP

We used three different scenarios for the three modeling tasks that were used in EXP1. The scenarios are different in topic, but slightly different in size.

In order to statistically assess the difference in the size, we calculated the number of *classes* and *associations* in each solution created by the students

Table 6.5: Number of classes and associations in the solutions of each modeling scenario

Scenario	**N of classes**				**N of associations**			
	Med.	Q1	Q3	I-Q. R.	Med.	Q1	Q3	I-Q. R.
1	8.00	7.00	8.00	1.00	7.00	6.00	7.00	1.00
2	6.00	4.00	9.00	5.00	5.00	3.00	8.00	5.00
3	6.50	6.00	7.75	1.75	5.50	5.00	6.75	1.75

per scenario. After that, we ran Kruskal-Wallis test [116]. The following two hypotheses were formed:

- Null Hypothesis H_0: There is no statistically significant difference between the median number of classes/associations in each solution created for scenario 1, 2 & 3.
- Alternative Hypothesis H_A: There is statistically significant difference between the median number of classes/associations in each solution created for scenario 1, 2 & 3.

The distributions of the number of classes and associations in the solutions of each modeling scenario are reported in Table 6.5. The result of Kruskal-Wallis test is presented in Table 6.4. We cannot reject the null hypothesis (the significance value $p = 0.126 > 0.05$). In other words, the difference between the number of classes/associations in each solution created for scenario 1, 2 and 3 is not significant.

Results for Q4: We cannot assess the impact of the size of the modeling task (scenario) on DEP, NEP and LEP, as we have evidence that the modeling tasks are not statistically different in size.

At this point we only study the impact of the topic of the modeling scenarios on DEP, NEEP and LEP. In particular, we want to asses if there is any statistically significant difference in the mean of DEP, NEEP and LEP between the three modeling scenarios. To this end, the following set of hypotheses were formed:

- Null Hypothesis H_{01}: There is no statistically significant difference in DEP of the three scenarios.
- Alternative Hypothesis H_{A1}: There is statistically significant difference in DEP of the three scenarios.

Table 6.6: Normality Test results

Data	**Shapiro-Wilk (p-value)**
DEP	0.046
NEEP	0.027
LEP	0.000

Table 6.7: Impact of the topic of the scenario on DEP, NEEP and LEP (Kruskal-Wallis test)

Data	**Chi-square**	**df**	**p-value**
DEP	1.040	2	0.595
NEEP	1.630	2	0.443
LEP	4.325	2	0.115

- Null Hypothesis H_{02}: There is no statistically significant difference in NEEP of the three scenarios.

- Alternative Hypothesis H_{A2}: There is statistically significant difference in NEEP of the three scenarios.

- Null Hypothesis H_{03}: There is no statistically significant difference in LEP of the three scenarios.

- Alternative Hypothesis H_{A3}: There is statistically significant difference in LEP of the three scenarios.

The normalities of DEP, NEEP and LEP were checked using Shapiro-Wilk test [117]. Table 6.6 shows the p-values of the test. The p-values are less than 0.05, and the data are not normally distributed.

Having non-normally distributed data, we applied the *non-parametric* Kruskal-Wallis test [116]. Table 6.7 shows the results of the test in detail. Since the significance values of DEP, NEEP and LEP are > 0.05, we cannot reject the null hypotheses. In other words, the differences between the mean rankings of DEP, NEEP and LEP of the given three modeling scenarios are not significant.

Furthermore, we compared the differences in the mean value of DEP, NEEP and LEP between every pair of scenarios using Mann-Whitney test [100]. The results reported in Table 6.8 prove that the difference in the mean rankings of DEP, NEEP and LEP is not significant between every pair of the three scenarios (the critical level of significance is $0.05/3 = 0.0167$).

Table 6.8: Impact of topic between every pair of scenarios (Mann-Whitney test)

	DEP (sig. 2-tailed)	**NEEP** (sig. 2-tailed)	**LEP** (sig. 2-tailed)
S1 - S2	0.408	0.316	0.168
S1 - S3	0.343	0.244	0.046
S2 - S3	0.949	0.733	0.394

Table 6.9: Obtained feedback via the questionnaire

Feedback	**Results**				
	Experiment	Med.	Q1	Q3	I-Q. R.
Expertise in software modeling	EXP1	2	1	3	2
	EXP2	3	2	3	1
Experience in using UML	EXP1	2	1	2	1
	EXP2	2	2	3	1
Clarity of the scenarios	EXP1	4	4	4	0
	EXP2	4	3	4	1
Usability of the modeling tool	EXP1	4	3	4	1
	EXP2	3	2	3	1

Results for Q5: We have statistical evidence to conclude that the DEP, NEEP and LEP stay the same through different-in-topic modeling tasks.

6.5.4 Subjects Questionnaire

This subsection resumes the feedback gathered from the participants involved in the two experiments. These feedback (presented in Table 6.9) complement the results, and are discussed in Sections 6.6 and 6.7.3.

6.6 Discussion

With the increasing popularity of agile-approaches in software development there has been a reduced commitment from software development projects to modeling. Partially this view seems motivated by the seeing of modeling as an activity that produces 'documentation' (rather than 'working code'). Our study shows that a significant part of the effort dedicated to modeling is spent on thinking about the design. Even though the actual impact of this needs to be further assessed, we believe that this thinking about the design is valuable.

For EXP1, our results show that *at least* 56% of the modeling effort is spent on *design*. Whereas for EXP2, *at least* 74% of the modeling effort is *design*. Our assumption is that the participants did not make any *design* effort while expressing the model in UML notation as well as doing layout.

One threat to the interpretation of our experiment is whether or not design-thinking actually happens concurrently with notation expression or layouting. Given the small size of the layout effort, this would only have a small impact on our interpretation. If one believes these cognitive tasks overlap, then the interpretation of our experiment should be that there is indeed more design thinking - i.e. we have found a lower bound on it through our study. Consequently, the percentages that we found are minimum, and may increase as the effort on *designing* overlaps with the effort on the other two activities (notation expression and layouting).

The *notation expression* effort is on average 41% and *layouting* is on average 3%. These efforts may actually represent a cost of UML modeling. This cost may be seen as an investment in the communication-value of documentation within a team. In order to understand whether the cost of notation expression and layouting is inevitable or not, we may investigate the impact of the usability of modeling tools on the modeling process. Although the participants of EXP1 used a modeling tool (WebUML) different from the one used in EXP2 (Papyrus), we could not reveal the impact of the modeling tool on the modeling effort. This is because the modeling solution of scenario 2 of EXP1 was created by participants individually, while the same scenario was modeled in by 4 participants collaborating in a team in EXP2. So, the difference in the DE, NEE and LE between the two settings could be due to the type of the modeling tool or the number of assigned participants per modeling task. Observing the perceived usability of the modeling tools by the participants of the two experiments (see Table 6.9), it might be that the difference in DE and NEE between EXP1 and EXP2 is due to the factor of collaboration, i.e. design discussions. As a future work, we aim to isolate the impact of these two factors (modeling tool and collaboration) on the modeling effort, as well as investigate better modeling-tool support [88].

6.7 Threats to Validity

6.7.1 Construct Validity

We benefited from the fact that we performed the experiment in a controlled environment (instead of as a homework assignment): 8 students created the model and its copy in parallel. We eliminated these cases from our data set because we could not explicitly calculate the modeling and copying efforts. For replication of this research, a lesson learned is to instruct students not to do *model* and *copy* simultaneously.

We did not aim at maximizing realism and focused on class diagrams for various reason. We wanted to: (i) ask simple tasks based on a well-known notation; (ii) reduce confounding factors and thus keep more control over the experiment compared to drawing multiple types of diagrams; and (iii) have a preliminary result that can validate our vision.

6.7.2 Internal Validity

It could be possible that thoughts wander off into unrelated territories. The following reasons limit the impact of this phenomenon: Firstly, the experiment was performed in a controlled environment. Supervisors walked around the room. They monitorred that there were no distractions like coffee drinking or going to the bathroom. Also, they observed that the participants were indeed working on the task behind their computers. Suppose that indeed some wandering of thoughts happens, then our expectation is that this happens more or less equally in the first phase (*modeling*) and the second phase (*copying*). In this case at least the relative ratio between these tasks should not be affected much.

In the redrawing of a copy of the previously created solution, participants in the study may have benefited from a *learning effect.* This could have led to the *notation expression effort* being a less than the *notation expression effort* in the initial creation of the UML models. We think this affect is small, and that a more detailed analysis of notation creation activities can lead to a quantification of this learning effect. However in *EXP2*, we tried to mitigate this factor by asking each group to copy the created solution of another group.

Moreover when performing the second modeling task (second scenario) in EXP1, participants may have benefited from a *learning effect* as well as suffered from the *fatigue* of performing two modeling tasks consecutively. We do not think there is a large *learning effect* in the use of the tool, because participants have been trained in using the tool both before starting and as part of this assignment. Hence they already have a reasonable fluency in the tool at the start of the first scenario. We do think the affect of *fatigue* is small because the modeling scenarios are simple, and the required time to complete one scenario is not too much (around 25 minutes).

6.7.3 External Validity

Complexity of the scenarios. The scenarios were kept simple and clear so that the students can easily understand and complete the tasks in the time of the experiment. In industry settings, modeling tasks can be much more complicated in term of size, terminology, languages, level of details, etc. which we could not cover in our study. This is a limitation on the generalizability of the findings of this paper when it comes to real-world cases. However, we consider this threat as acceptable for this preliminary investigation. We are working on studying larger scenarios to increase the generalizability.

Participants and their modeling expertise. The participants involved in our experiment may not represent the general population of modeling practitioners. Moreover, the modeling expertise of our participants is relatively homogeneous. This limits us from generalizing our findings to other subjects (i.e. experts, professional software architects, industrial practitioners in the field). Indeed, familiarity with the modeling tool and experience of designing may result in different DE, NEE and LE percentages. We consider our findings as a basis to extend our study to larger community of modeling practitioners.

6.8 Conclusion and Future Work

In order to better understand the effort involved in using software models in software development, we introduced in this paper the distinction between *design*, *notation expression* and *layout* efforts. Subsequently, we defined and ran two experiments in which we measure how much effort each of these activities takes – both in absolute effort and as a percentage of the total effort spent on creating class diagrams in a simple student assignment. From these experiments we conclude that UML modeling should not be considered so very costly, because it triggers design thinking. According to our results, the effort spent on thinking about synthesizing the design takes *at least* 56% of the total modeling effort.

One implication of this research is that projects that create models concur at least with significant thinking about the design. This aligns with an earlier finding that developers report that creating design models in the early stage of a software development projects, leads to better modularity of the design [80].

Future work: This line of research can be extended in many directions. In order to increase the external validity, we would like to obtain data from larger and/or industrial projects about their modeling effort. We have started looking into two projects where teams of 8 students work for 3 months full time on a software project. Another extension that is of interest is to study the effect of usability of modeling tools on the time spent on subtasks of modeling. This would highlight if tool complexity is a major factor in adoption of modeling. Complementary questions would be to: (i) explore how much effort is involved in maintaining models up-to-date in documentation throughout a project, and (ii) study the impact of the *designing* and *modeling* on the speed and quality of software development.

Bibliography

[1] B. Selic, "The pragmatics of model-driven development," *IEEE software*, vol. 20, no. 5, pp. 19–25, 2003.

[2] M. Cherubini, G. Venolia, R. DeLine, and A. J. Ko, "Let's go to the whiteboard: how and why software developers use drawings," in *Proceedings of the SIGCHI conference on Human factors in computing systems.* ACM, 2007, pp. 557–566.

[3] K. Lyytinen and V.-P. Tahvanainen, *Next generation CASE tools.* IOS Press, 1992, vol. 3.

[4] D. Budgen, "The cobbler's children: Why do software design environments not support design practices?" in *Software Designers in Action: A Human-Centric Look at Design Work.* Chapman and Hall/CRC, 2013, pp. 199–216.

[5] S. Abrahão, R. F. Paige, S. Kokaly, B. Cheng, F. Bordeleau, H. Störrle, and J. Whittle, "User experience for model-driven engineering: Challenges and future directions," in *ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems*, 2017.

[6] N. L. Chervany and D. Lending, "Case tools: understanding the reasons for non-use," *ACM SIGCPR Computer Personnel*, vol. 19, no. 2, pp. 13–26, 1998.

[7] L. Fowler, J. Armarego, and M. Allen, "Case tools: Constructivism and its application to learning and usability of software engineering tools," *Computer Science Education*, vol. 11, no. 3, pp. 261–272, 2001.

[8] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "A taxonomy of tool-related issues affecting the adoption of model-driven engineering," *Software & Systems Modeling*, vol. 16, no. 2, pp. 313–331, 2017.

[9] I. Hammouda, H. Burden, R. Heldal, and M. R. Chaudron, "Case tools versus pencil and paper: A student's perspective on modeling software design." in *EduSymp@ MoDELS*, 2014, pp. 21–30.

[10] S. Baltes and S. Diehl, "Sketches and diagrams in practice," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering.* ACM, 2014, pp. 530–541.

[11] F. Tomassetti, M. Torchiano, A. Tiso, F. Ricca, and G. Reggio, "Maturity of software modelling and model driven engineering: A survey in the italian industry," 2012.

[12] B. Moggridge and B. Atkinson, *Designing interactions.* MIT press Cambridge, MA, 2007, vol. 17.

[13] P. Cohen and S. Oviatt, "Multimodal interfaces that process what comes naturally," *Commun ACM*, vol. 43, no. 3, pp. 45–33, 2000.

[14] J. D. Herbsleb, "Global software engineering: The future of socio-technical coordination," in *2007 Future of Software Engineering.* IEEE Computer Society, 2007, pp. 188–198.

[15] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, "Empirical assessment of MDE in industry," in *Proceedings of the 33rd Int. Conf. on Software Engineering.* ACM, 2011, pp. 471–480.

[16] P. Mohagheghi and V. Dehlen, "Where is the proof?–a review of experiences from applying MDE in industry," *Lecture Notes in Computer Science*, vol. 5095, no. 2008, pp. 432–443, 2008.

[17] T. Gorschek, E. Tempero, and L. Angelis, "On the use of software design models in software development practice: An empirical investigation," *Journal of Systems and Software*, vol. 95, pp. 176–193, 2014.

[18] M. Petre, "Uml in practice," in *Proceedings of the 2013 International Conference on Software Engineering.* IEEE Press, 2013, pp. 722–731.

[19] P. Ralph and Y. Wand, "A proposal for a formal definition of the design concept," *Design requirements engineering: A ten-year perspective*, vol. 14, pp. 103–136, 2009.

[20] P. Bourque, R. E. Fairley *et al.*, *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0.* IEEE Computer Society Press, 2014.

[21] D. Budgen, *Software design.* Pearson Education, 2003.

[22] H. W. Rittel and M. M. Webber, "Dilemmas in a general theory of planning," *Policy sciences*, vol. 4, no. 2, pp. 155–169, 1973.

[23] J. Kramer and O. Hazzan, "The role of abstraction in software engineering," in *Proceedings of the 28th international conference on Software engineering.* ACM, 2006, pp. 1017–1018.

[24] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[25] D. D. Chamberlin and R. F. Boyce, "Sequel: A structured english query language," in *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control.* ACM, 1974, pp. 249–264.

[26] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis Lectures on Software Engineering*, vol. 1, no. 1, pp. 1–182, 2012.

[27] S. Kent, "Model driven engineering," in *Integrated formal methods.* Springer, 2002, pp. 286–298.

[28] S. J. Mellor, T. Clark, and T. Futagami, "Model-driven development: guest editors' introduction." *IEEE software*, vol. 20, no. 5, pp. 14–18, 2003.

[29] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice," *Software & Systems Modeling*, pp. 1–23, 2016.

[30] P. Baker, S. Loh, and F. Weil, "Model-driven engineering in a large industrial contextmotorola case study," *Model Driven Engineering Languages and Systems*, pp. 476–491, 2005.

[31] M. Petre and A. Van Der Hoek, *Software Designers in Action: A Human-Centric Look at Design Work.* CRC Press, 2013.

[32] A. Tang, A. Aleti, J. Burge, and H. van Vliet, "What makes software design effective?" *Design Studies*, vol. 31, no. 6, pp. 614–640, 2010.

[33] B. Sharif, N. Dragan, A. Sutton, M. L. Collard, and J. I. Maletic, "Identifying and analyzing software design activities," in *Software Designers in Action: A Human-Centric Look at Design Work.* Chapman and Hall/CRC, 2013, pp. 153–174.

[34] A. Baker and A. van der Hoek, "Ideas, subjects, and cycles as lenses for understanding the software design process," *Design Studies*, vol. 31, no. 6, pp. 590–613, 2010.

[35] U. Dekel and J. D. Herbsleb, "Notation and representation in collaborative object-oriented design: an observational study," in *ACM SIGPLAN Notices*, vol. 42, no. 10. ACM, 2007, pp. 261–280.

[36] R. Wieringa, *Design Science Methodology for Information Systems and Software Engineering.* Springer, 2014.

[37] C. R. Kothari, *Research methodology: Methods and techniques.* New Age International, 2004.

[38] J. D. Gould and C. Lewis, "Designing for usability: key principles and what designers think," *Communications of the ACM*, vol. 28, no. 3, pp. 300–311, 1985.

[39] H. Sharp, Y. Rogers, and J. Preece, "Interaction design: beyond human-computer interaction," 2007.

[40] R. Kosara, C. G. Healey, V. Interrante, D. H. Laidlaw, and C. Ware, "Thoughts on user studies: Why, how, and when," *IEEE Computer Graphics and Applications*, vol. 23, no. 4, pp. 20–25, 2003.

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

[42] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, "Selecting empirical methods for software engineering research," *Guide to advanced empirical software engineering*, pp. 285–311, 2008.

[43] J. Sauro, *A practical guide to the system usability scale: Background, benchmarks & best practices.* Measuring Usability LLC, 2011.

[44] P. D. Cherulnik, *Methods for behavioral research: A systematic approach.* Sage Publications, 2001.

[45] C. U. Smith and L. G. Williams, "Performance solutions: a practical guide to creating responsive, scalable software," 2001.

[46] J. L. Fiadeiro, "The many faces of complexity in software design," in *Conquering Complexity.* Springer, 2012, pp. 3–47.

[47] A. Filieri, C. Ghezzi, R. Mirandola, and G. Tamburrelli, "Conquering complexity via seamless integration of design-time and run-time verification," in *Conquering Complexity.* Springer, 2012, pp. 253–275.

[48] E. Lank, J. Thorley, S. Chen, and D. Blostein, "On-line recognition of uml diagrams," in *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on.* IEEE, 2001, pp. 356–360.

[49] C. H. Damm, K. M. Hansen, and M. Thomsen, "Tool support for cooperative object-oriented design: gesture based modelling on an electronic whiteboard," in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems.* ACM, 2000, pp. 518–525.

[50] B. Plimmer and M. Apperley, "Computer-aided sketching to capture preliminary design," in *Australian Computer Science Communications*, vol. 24, no. 4. Australian Computer Society, Inc., 2002, pp. 9–12.

[51] N. Mangano, T. D. LaToza, M. Petre, and A. van der Hoek, "Supporting informal design with interactive whiteboards," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* ACM, 2014, pp. 331–340.

[52] D. Wüest, N. Seyff, and M. Glinz, "Flexisketch: A mobile sketching tool for software modeling," *Mobile Computing, Applications, and Services*, vol. 110, pp. 225–244, 2013.

[53] ——, "Flexisketch team: Collaborative sketching and notation creation on the fly," in *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, vol. 2. IEEE, 2015, pp. 685–688.

[54] Q. Chen, J. Grundy, and J. Hosking, "An e-whiteboard application to support early design-stage sketching of uml diagrams," in *Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on.* IEEE, 2003, pp. 219–226.

[55] J. Grundy and J. Hosking, "Supporting generic sketching-based input of diagrams in a domain-specific visual language meta-tool," in *Proceedings of the 29th international conference on Software Engineering.* IEEE Computer Society, 2007, pp. 282–291.

[56] M. Magin and S. Kopf, "A collaborative multi-touch uml design tool," *Technical reports*, vol. 13, 2013.

[57] "Open Services for Lifecycle Collaboration OSLC, an open community building practical specifications for integrating software," http://open-services.net/, accessed: 2015-07-16.

[58] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, and M. Wimmer, "An introduction to model versioning," in *Proceedings of the 12th international conference on Formal Methods for the Design of Computer, Communication, and Software Systems: formal methods for model-driven engineering.* Springer-Verlag, 2012, pp. 336–398.

[59] K. Altmanninger, M. Seidl, and M. Wimmer, “A survey on model versioning approaches,” *International Journal of Web Information Systems*, vol. 5, no. 3, pp. 271–304, 2009.

[60] “IBM Rational Development Family,” http://www-03.ibm.com/software/products/en/developer, accessed: 2015-09-03.

[61] T. R. G. Green and M. Petre, “Usability analysis of visual programming environments: a cognitive dimensions framework,” *Journal of Visual Languages & Computing*, vol. 7, no. 2, pp. 131–174, 1996.

[62] D. Moody, “The physics of notations: toward a scientific basis for constructing visual notations in software engineering,” *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009.

[63] S. J. Hoppenbrouwers, H. Proper, and T. P. van der Weide, “Formal modelling as a grounded conversation,” 2005.

[64] B. Nuseibeh, “Weaving together requirements and architectures,” *Computer*, vol. 34, no. 3, pp. 115–119, 2001.

[65] K. Nakakoji, Y. Yamamoto, N. Matsubara, and Y. Shirai, “Toward unweaving streams of thought for reflection in professional software design,” *IEEE software*, vol. 29, no. 1, pp. 34–38, 2012.

[66] M. R. Chaudron, W. Heijstek, and A. Nugroho, “How effective is uml modeling?” *Software & Systems Modeling*, vol. 11, no. 4, pp. 571–580, 2012.

[67] B. Tversky, “Visualizing thought,” *Topics in Cognitive Science*, vol. 3, no. 3, pp. 499–535, 2011.

[68] J. Walny, S. Carpendale, N. H. Riche, G. Venolia, and P. Fawcett, “Visual thinking in action: Visualizations as used on whiteboards,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 12, pp. 2508–2517, 2011.

[69] B. Tversky, “What do sketches say about thinking,” in *2002 AAAI Spring Symposium, Sketch Understanding Workshop, Stanford University, AAAI Technical Report SS-02-08*, 2002, pp. 148–151.

[70] N. Mangano, T. D. LaToza, M. Petre, and A. Van Der Hoek, “How software designers interact with sketches at the whiteboard,” *Software Engineering, IEEE Transactions on*, vol. 41, no. 2, pp. 135–156, 2015.

[71] M. R. Chaudron and R. Jolak, “A vision on a new generation of software design environments,” in *First Int. Workshop on Human Factors in Modeling (HuFaMo 2015). CEUR-WS*, 2015, pp. 11–16.

[72] R. Jolak, B. Vesin, and M. R. V. Chaudron, "OctoUML: An environment for exploratory and collaborative software design," in *Proceedings of 39th Int. Conference on Software Engineering. ICSE'17*, 2017, pp. 7–10.

[73] J. Iivari, "Why are case tools not used?" *Communications of the ACM*, vol. 39, no. 10, pp. 94–103, 1996.

[74] D. Wüest, N. Seyff, and M. Glinz, "Flexisketch: A mobile sketching tool for software modeling," in *Mobile Computing, Applications, and Services*. Springer, 2012, pp. 225–244.

[75] J. Walny, J. Haber, M. Dörk, J. Sillito, and S. Carpendale, "Follow that sketch: Lifecycles of diagrams and sketches in software development," in *Visualizing Software for Understanding and Analysis (VISSOFT), 2011 6th IEEE International Workshop on.* IEEE, 2011, pp. 1–8.

[76] B. Paulson and T. Hammond, "Paleosketch: accurate primitive sketch recognition and beautification," in *Proceedings of the 13th international conference on Intelligent user interfaces.* ACM, 2008, pp. 1–10.

[77] J. Brooke *et al.*, "Sus-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[78] J. Corbin and A. Strauss, *Basics of qualitative research: Techniques and procedures for developing grounded theory.* Sage publications, 2014.

[79] T. S. Tullis and J. N. Stetson, "A comparison of questionnaires for assessing website usability," in *Usability Professional Association Conference*, 2004, pp. 1–12.

[80] A. Nugroho and M. R. Chaudron, "A survey into the rigor of uml use and its perceived impact on quality and productivity," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement.* ACM, 2008, pp. 90–99.

[81] S. Lahtinen and J. Peltonen, "Adding speech recognition support to uml tools," *Journal of Visual Languages & Computing*, vol. 16, no. 1, pp. 85–118, 2005.

[82] M. B. Albizuri-Romero, "A retrospective view of case tools adoption," *ACM SIGSOFT Software Engineering Notes*, vol. 25, no. 2, pp. 46–50, 2000.

[83] R. A. Harris, *Voice interaction design: crafting the new conversational speech systems.* Elsevier, 2004.

[84] A. E. Lackey, T. Pandey, M. Moshiri, N. Lalwani, C. Lall, and P. Bhargava, "Productivity, part 2: cloud storage, remote meeting tools, screencasting, speech recognition software, password managers, and online data backup," *Journal of the American College of Radiology*, vol. 11, no. 6, pp. 580–588, 2014.

[85] A. Drigas and G. Papanastasiou, "Interactive white boards in preschool and primary education." *iJOE*, vol. 10, no. 4, pp. 46–51, 2014.

[86] F. Gursul and G. B. Tozmaz, "Which one is smarter? teacher or board," *Procedia-Social and Behavioral Sciences*, vol. 2, no. 2, pp. 5731–5737, 2010.

[87] Å. Fast-Berglund, U. Harlin, and M. Åkerman, "Digitalisation of meetings–from white-boards to smart-boards," *Procedia CIRP*, vol. 41, pp. 1125–1130, 2016.

[88] R. Jolak, B. Vesin, M. Isaksson, and M. R. Chaudron, "Towards a new generation of software design environments: Supporting the use of informal and formal notations with octouml," in *Second International Workshop on Human Factors in Modeling (HuFaMo 2016). CEUR-WS*, 2016, pp. 3–10.

[89] J. Larson, S. Oviatt, and D. Ferro, "Designing the user interface for pen and speech applications," in *CHI'99 Workshop, Conference on Human Factors in Computing Systems (CHI'99)*, 1999.

[90] S. Oviatt, P. Cohen, L. Wu, L. Duncan, B. Suhm, J. Bers, T. Holzman, T. Winograd, J. Landay, J. Larson *et al.*, "Designing the user interface for multimodal speech and pen-based gesture applications: state-of-the-art systems and future research directions," *Human-computer interaction*, vol. 15, no. 4, pp. 263–322, 2000.

[91] M. Van Tulder, A. Malmivaara, and B. Koes, "Repetitive strain injury," *The Lancet*, vol. 369, no. 9575, pp. 1815–1822, 2007.

[92] S. C. Arnold, L. Mark, and J. Goldthwaite, "Programming by voice, vocalprogramming," in *Proceedings of the fourth international ACM conference on Assistive technologies.* ACM, 2000, pp. 149–155.

[93] A. Begel, "Spoken language support for software development," in *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on.* IEEE, 2004, pp. 271–272.

[94] T. J. Hubbell, D. D. Langan, and T. F. Hain, "A voice-activated syntax-directed editor for manually disabled programmers," in *Proceedings of*

the 8th International ACM SIGACCESS Conference on Computers and Accessibility. ACM, 2006, pp. 205–212.

[95] S. Mills, S. Saadat, and D. Whiting, "Is voice recognition the solution to keyboard-based rsi?" in *2006 World Automation Congress.* IEEE, 2006, pp. 1–6.

[96] F. Soares, J. Araújo, and F. Wanderley, "Voicetomodel: an approach to generate requirements models from speech recognition mechanisms," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing.* ACM, 2015, pp. 1350–1357.

[97] T. Nishimoto, N. Shida, T. Koayashi, and K. Shirai, "improving human interface drawing tool using speech, mouse and key-board," in *Robot and Human Communication, 1995. RO-MAN'95 TOKYO, Proceedings., 4th IEEE International Workshop on.* IEEE, 1995, pp. 107–112.

[98] L. Rabiner and B. Juang, "An introduction to hidden markov models," *IEEE ASSP magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[99] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf, and J. Woelfel, "Sphinx-4: A flexible open source framework for speech recognition," 2004.

[100] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[101] S. Oviatt and P. Cohen, "Perceptual user interfaces: multimodal interfaces that process what comes naturally," *Communications of the ACM*, vol. 43, no. 3, pp. 45–53, 2000.

[102] R. Jolak, B. Vesin, M. Isaksson, and M. R. V. Chaudron, "Towards a new generation of software design environments: Supporting the use of informal and formal notations with octouml," in *Second Int. Workshop on Human Factors in Modeling. CEUR-WS*, 2016, p. : in print.

[103] B. Selic, "What will it take? a view on adoption of model-based methods in practice," *Software & Systems Modeling*, vol. 11, no. 4, pp. 513–526, 2012.

[104] B. Anda, K. Hansen, I. Gullesen, and H. K. Thorsen, "Experiences from introducing UML-based development in a large safety-critical project," *Empirical Software Engineering*, vol. 11, no. 4, pp. 555–581, 2006.

[105] D. Budgen, A. J. Burn, O. P. Brereton, B. A. Kitchenham, and R. Pretorius, "Empirical evidence about the UML: a systematic literature review," *Software: Practice and Experience*, vol. 41, no. 4, pp. 363–392, 2011.

[106] W. Mischel, E. B. Ebbesen, and A. Raskoff Zeiss, "Cognitive and attentional mechanisms in delay of gratification." *Journal of personality and social psychology*, vol. 21, no. 2, p. 204, 1972.

[107] P. Devanbu, T. Zimmermann, and C. Bird, "Belief and evidence in empirical software engineering," in *Proceedings of the International Conference on Software Engineering*. ACM, 2016, p. in print.

[108] D. Stikkolorum, T. Ho-Quang, and M. R. V. Chaudron, "Revealing students' UML class diagram modelling strategies with webuml and logviz," in *SEAA, 2015 41st Euromicro*. IEEE, 2015, pp. 275–279.

[109] D. R. Stikkolorum, T. Ho-Quang, B. Karasneh, and M. R. Chaudron, "Uncovering students' common difficulties and strategies during a class diagram design process: an online experiment." in *EduSymp@ MoDELS*, 2015, pp. 29–42.

[110] M. Brambilla and P. Fraternali, "Large-scale model-driven engineering of web user interaction: The WebML and WebRatio experience," *Science of Computer Programming*, vol. 89, Part B, pp. 71 – 87, 2014, special issue on Success Stories in Model Driven Engineering.

[111] O. Diaz and F. M. Villoria, "Generating blogs out of product catalogues: An MDE approach," *Journal of Systems and Software*, vol. 83, no. 10, pp. 1970 – 1982, 2010.

[112] M. Brambilla, A. Mauri, and E. Umuhoza, "Extending the Interaction Flow Modeling Language (IFML) for Model Driven Development of Mobile Applications Front End," in *MobiWIS*, 2014, pp. 176–191.

[113] D. Klyve and L. Stemkoski, "Graeco-Latin Squares and a Mistaken Conjecture of Euler," *The College Mathematics Journal*, vol. 37, no. 1, 2006.

[114] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013, ISBN 3-900051-07-0. [Online]. Available: www.R-project.org

[115] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and psychological measurement*, vol. 20, no. 1, pp. 37–46, 1960.

[116] W. H. Kruskal and W. A. Wallis, “Use of ranks in one-criterion variance analysis,” *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[117] S. S. Shapiro and M. B. Wilk, “An analysis of variance test for normality (complete samples),” *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.