

CONCEPT MAP BASED
SOFTWARE ENGINEERING

A thesis submitted by

Thomas Hubbard

In partial fulfillment of the requirements
for the degree of

Master of Science

in

Computer Science

TUFTS UNIVERSITY

May, 2007

© 2007, Thomas Hubbard

ADVISOR:

Judith Stafford

UMI Number: 1442521

Copyright 2007 by
Hubbard, Thomas

All rights reserved.

UMI[®]

UMI Microform 1442521

Copyright 2007 by ProQuest Information and Learning Company.
All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

This thesis proposes that a concept map be used to support a shared understanding of a software system. The concept map will be used to provide a common context to be used throughout the software development lifecycle. This common context would remove the problem of requiring technical views of the system be translated to generic, ambiguous diagrams for the purposes for stakeholder communications. In addition, since this shared conceptual model of the system will act as a context for many other conceptual models then it can also act as a bridge to map between views of a system thereby enabling different views of the system to be cross referenced to each other using a common context.

The approach is examined using a pilot study, the development of a concept map tutorial and the creation of a partial mapping between Unified Modeling Language notation and concept map linking words.

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to my thesis advisor, Dr. Judith Stafford for her support and encouragement. I am indebted to her for her ability to make succinct what has often been presented to her in a rambling fashion. She has helped guide and focus me on core aspects and side effects of the problem of a lack of a shared understanding between shareholders. She was instrumental in directing the work into mapping between views of a system using the concept map. In addition I would like to thank David Kahle for his direction and guidance into the field of participatory design a subject that, without his input, I would not been aware of which would have been detrimental to this work. Participatory design has the same goals as the approach proposed in this work Mr. Kahle's guidance provided me with valuable insight into the larger problem of designing systems that takes into account the social environment that the designers must operate in. I am grateful to my advisor, Professor Robert Jacob for his continued support in providing me considerable latitude in my research activities. I would also like to show my appreciation to the designers of the Visual Understanding Environment (VUE) for all their hard work. The tool that they have created has provided me with considerable value over the last two years in the creation of my concept maps. Finally, I would like to thank my wife and children for their continued support and understanding during this, often times, difficult journey.

TABLE OF CONTENTS

Acknowledgments	iv
Table of Contents	1
List of Tables	2
List of Figures	3
Introduction.....	4
Introduction to Communication Issues in Software Engineering	7
Introduction to Concept Maps	11
Proposed Solution	14
Methodology	16
Results.....	17
Conclusion	34
Appendix A Concept Map Based Software Engineering Tutorial.....	36
Bibliography.....	51

LIST OF TABLES

Table 1 Partial Mapping of UML Notation to Concept Map Connecting Words	26
Table 2 Model Mapping.....	31

LIST OF FIGURES

<i>Number</i>	<i>Page</i>
Figure 1 Sample Architecture Views [Clements].....	9
Figure 2 Conceptual Model Translation from a Module View in UML to a Customer-friendly C&C view.	10
Figure 3 Concept Map Example [Novak 1998].....	12
Figure 4 Team 3 Concept Map Showing Enrollment.....	20
Figure 5 Team 3 Concept Map Showing Discussions and Forms.....	21
Figure 6 Team 3 “Boxes and lines” diagram.....	21
Figure 7 Team 4 Concept Map.....	23
Figure 9 Team 1’s UML Based Concept Map.....	24
Figure 10 Partial view of Team 3’s Use Case Diagram.....	28
Figure 11 Team 3 Concept Maps Merged Together.....	29
Figure 12 Mapping Team 3’s Models.....	31

INTRODUCTION

Software is created to fulfill the needs of a user. The needs of the user are typically defined and presented in natural language requirements documents, use cases or developed as prototypes. However, unlike most products, software takes form as a conceptual model of reality. There are physical representations of the conceptual model, source code documents, requirements documents, users manuals, executing processes, media with compiled executables and the like but the fundamental essence of software is an intangible conceptual model. System architects use specialized modeling languages such as the Unified Modeling Language (UML) to give this abstract model a form that can be used in discussions and decision making. When an abstract notion is modeled using a modeling language and then communicated with other like-minded system architects the same general conceptual model is often envisioned by the participants. This process is much like when a person is asked to think of a house. Each person has the same basic model formed in their mind's eye, kitchen, bed room, bathroom and a roof, yet each person's interpretation is slightly different, town home, ranch, split level, et cetera. In other words, just because the model is the same doesn't mean our vision is the same.

Imagine if someone only had a rudimentary knowledge of the modeling language used to represent a conceptual model of a house and we asked if the house presented in the floor plans, electrical plans and plumbing plans met their needs. Unfortunately system architects ask stakeholders these questions every day. Consider a contractor building houses and only showing clients floor plans. Based on these floor plans the client must agree or disagree that this is the house they need. To compensate for a lack of understanding, stakeholders will trust that the people building the house understood what they wanted and will let them continue building. Once the house is complete only then will the client discover that this truly is an ugly house that does not meet any of their needs. At this point the contractor can then either fix the problem or contend that the client agreed to the floor plan and therefore they got what they deserved.

Would this scenario ever happen in reality? No, because in reality homebuilders not only have floor plans but also have 3-D pictures of what the interior and exterior of the house will look like, a vision. The client can easily use this vision to see if their wants and needs are being met long before anything is built. In other words, the homebuilder and the client have a shared understanding of what is being proposed. Using this shared understanding all stakeholders can collectively move toward the same goal.

Turning our attention to the software development process consider the different artifacts creating during the different stages of software system development, the requirements document, the software architecture document, software design document, source code, schedules, et cetera [IEEE, EIA 1998] and their representation. Requirements documents are typically expressed in natural language text [Christel 1992], design and architecture documents are often represented in Unified Modeling Language (UML) and schedules are often represented using GANTT charts. The reason that there are so many representations for the documentation in a system is that each representation is superior to the others at a particular job. Natural language is used in requirements documents to avoid misunderstanding due to terminology differences [Christel 1992]. UML is superior to GANTT charts or natural language to describe a system architecture because imagery and standard notations reduce the complexity of the system being described [Knight 1998]. GANTT charts are superior to either natural language or UML due to the resource orientation and basis in time.

Due to the large number of different documents in use in the software engineering process and the specialized purpose they serve several conceptual models exist of the system. Since these documents drive the development process in essence the software development process is asking the client to agree or disagree that this is the house they need based on plumbing plans, electrical plans, floor plans, legal contracts, paint samples and project plans. Clearly a shared understanding of the system is required and a conceptual model to support such a shared understanding is missing. This claim is reinforced in the theory of Social Construction of Technology (SCOT). The SCOT framework places emphasis on the different social perspectives that drive the development of new technologies [Gay]. A primary assumption of the SCOT approach, as in concept map based software engineering, is that communication

between stakeholders is critical to coordinate versions of the design and components of the system [Gay].

This illustrates the fundamental problem of a lack of a common view or conceptual model that supports a shared understanding of a system. A second problem is the difficulty in mapping between views or conceptual models of a system. This work proposes that if a common view that supported a shared understanding is developed then it can be used to provide a context for other views of the system. In addition, since the shared view of the system acts a context for many views then it can also act as a bridge to map between views of a system thereby allowing different views of the system to be cross referenced to each other using a common context.

This work proposes a software engineering artifact based on the techniques of concept mapping be used to provide this shared understanding of the system by graphically illustrating the concepts identified in the requirements document with associated behavior and restrictions. This concept map will live throughout the lifecycle process and provide a common view of the system context, manage the volume of information and enable a common language for referencing components of the system. Rather than simply being an output of one phase of the software development process, the concept map would span multiple phases and provide a context from which stakeholders can derive a common terminology and understanding of the system¹.

¹ This process will be referred to in this thesis as *Concept Map Based Software Engineering*.

INTRODUCTION TO COMMUNICATION ISSUES IN SOFTWARE ENGINEERING

Returning to the discussion on software engineering, the artifacts created in software engineering e.g. the requirements document, design document and project plans are conceptual models of the system used by different stakeholders. Conceptual models provide the means by which all relevant aspects of a problem domain are recorded and communicated [Genero]. Each conceptual model has a set of structures and semantics from which the model can derive its expressiveness. Each model will record a subset of the aspects of the problem domain, other aspects of the domain will be marginalized and yet other aspects will not be able to be expressed in the model. Since a conceptual model cannot express all important aspects of a problem domain the model will act as filter of the problem domain [Genero]. Referring back to the artifacts in the software development process, each artifact is a conceptual model of the system, each with expressive capabilities and strengths. For example the project plan is a conceptual model of the system that is better suited to model the temporal aspects of the project than other models such as the requirements document.

Ideally all stakeholders that come into contact with a system would have a shared understanding of the system being developed by being intimately aware of the requirements document, the environmental constraints and all background information needed to be active project participants. Often times however compartmentalized understanding of the system being developed is maintained as part of a team consciousness with individual members becoming experts in parts of the system. According to Smolander this behavior hints at organizational problems and general management urges that this architectural knowledge to be spread throughout the organization at the risk of the architectural design and description being impeded between different stakeholder groups. To combat this several processes have been created to provide visibility into the system development process. One process which is heavily document intensive and is used to provide architectural visibility is the Department of Defense process named DoD-STD-2167A. Often documenting systems is often not enough because there are issues with documents themselves. Documents tend to become obsolete over time

and since design knowledge is contextual, the documents will have multiple views of the same aspects of the system which may cause the documents to become large and impenetrable [Cederling]. Finally, those documents which are written in natural language tend to suffer from ambiguities caused by circular references in the language itself. For example, one can easily look-up a word in a dictionary and follow the definitions of the subsequent words. It does not take many iterations before the original definition is reached again. For example, consider the word 'dimension' which can take the meanings 'measure', 'extent' and 'magnitude'. The words 'measure', 'extent' and 'magnitude' can be defined using the word 'size'. Finally the word 'size' is defined using the word 'dimension' thus completing the cycle [Preparata]. To illustrate the issue with regards to documentation and presentation: If the concept of 'dimension' was used in the system documentation but one stakeholder group were to use the concepts of 'dimension' and 'size' interchangeably then the other stakeholder groups would have to be cognizant of which group created the presentation or document to understand that whenever the word 'size' were encountered it should be translated to the common term 'dimension'.

View-based approaches to documentation are used to compensate for deficiencies in other approaches because view-based approaches are effective tools for documenting architectures [Clements]. However, view based approaches suffer from a lack of tools and techniques to effectively map between different views of the system. Documenting architectures such that different stakeholders can find the information they need and to understand the information once it is found is a difficult task. Some of the issues include creating a view of the system that provides the stakeholder a contextual view of the elements and interrelationships between those elements. Next, descriptions of systems are often presented to stakeholders using natural language documents or informal presentations. Views of the system are often presented in technical notation that many stakeholders cannot use effectively to perform their jobs [Smolander]. Finally, the runtime view of a system and the implementation view of a system often differ greatly leaving the reader unable to understand how elements relate to one and another.

In order to illustrate the differences between runtime view of a system and an implementation view a simple example is presented using current architectural views. Figure 1 is based heavily

on the Capitalize System from [Clements]. This simple system reads characters from an input stream such as stdin, copies the characters, and outputs characters to a new stream. The characters written to the output stream will be changed so that they have alternating case. In addition, the software system can be configured at compile time. The configuration information includes the source and sink of the character stream and the language of the characters in the streams.

Referring to Figure 1, the diagram on the left illustrates a Component and Connector (C&C) Viewtype of the system. The C&C view shows the runtime view of the same system.

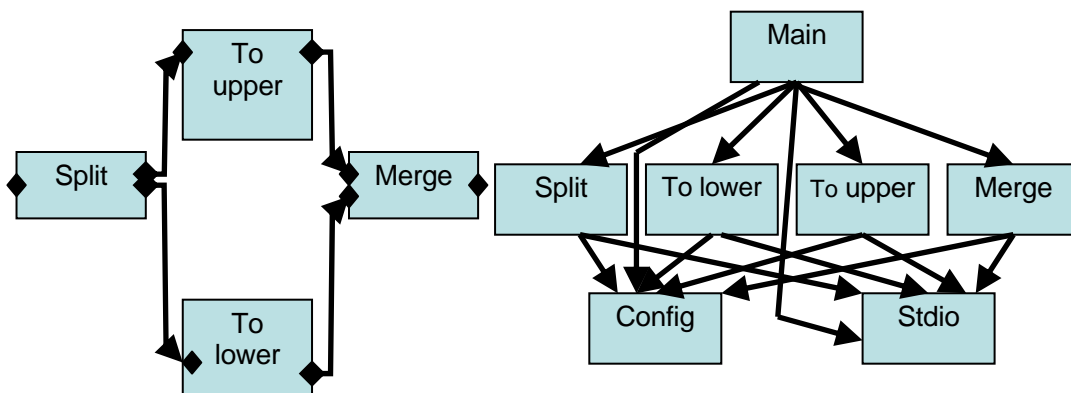


Figure 1 Sample Architecture Views [Clements]

Since the relationship between views of the architecture provide significant insights into the system it is necessary to create a document that maps between the views [Bass 2002]. Unfortunately these mappings can be difficult to create particularly when one code module maps to many runtime elements or when code modules are not instantiated as runtime elements.

Consider what occurs when several different stakeholder groups meet, the different conceptual models of the system must be translated from the conceptual model used by one group to a more general model for use with a broader audience. In this thesis, the activity of translating one stakeholder's model to another is known as *model translation*. An example of a model translation is shown in Figure 2. Consider two stakeholders that are meeting to determine if an

architecture will meet all the requirements imposed on the system. One stakeholder is the system architect and the other is the customer who ordered the system being built. The system architect is using UML to represent the system so she must translate the model into something the customer understands.

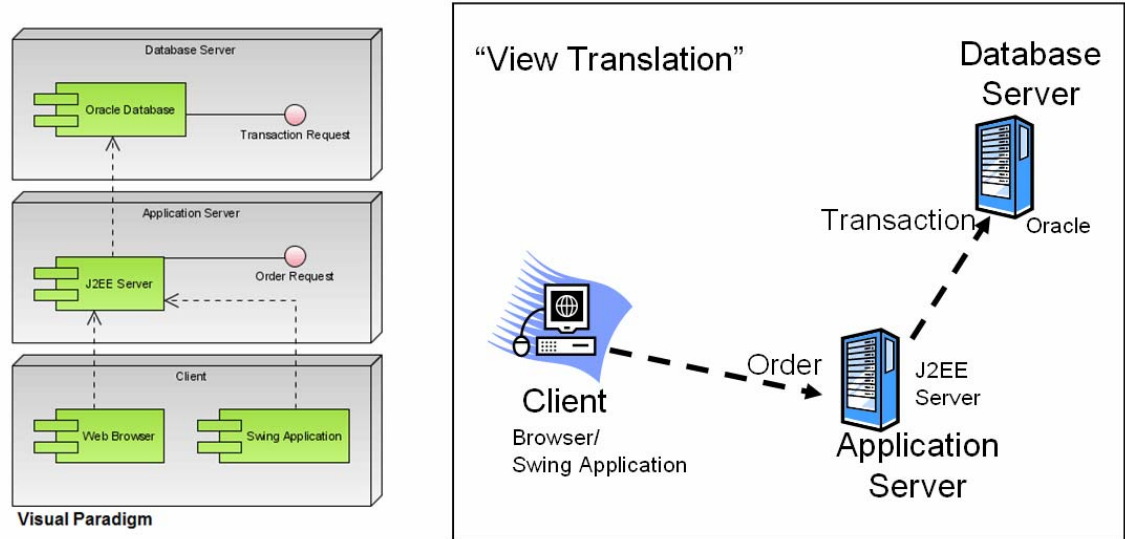


Figure 2 Conceptual Model Translation from a Module View in UML to a Customer-friendly C&C view.

Referring to Figure 2, the image on the left represents a module view of a three tier client server system in UML notation using packages. The image on the right represents a C&C view of the same system using a non-technical, customer friendly diagram generated using a business application such as Microsoft's PowerPoint® [Grinter].

The problem with model translations are that they are 1) inaccurate due to the subjective nature of the translation 2) inconsistent since the translations are done in an ad-hoc manner and therefore vary from presenter to presenter 3) time consuming to produce 4) inaccurate when translated from a more robust representation of the system and 5) out of date with reality since work must be done to keep the 'stakeholder' model in line with the working model. All of these problems result in miscommunications and wasted time and effort.

INTRODUCTION TO CONCEPT MAPS

The first step in the concept map based software engineering paradigm is the creation of a concept map. Since the idea of a concept can mean many things to many people certain common definitions must be agreed upon. Using the definition put forth by the creator of concept maps, Professor Joseph D. Novak of Cornell University, a concept is a “perceived regularity in events or objects, or records of events, designated by a label” [Novak]. It should be noted that since meaning is context dependent the label associated with a concept will have some idiosyncratic elements because no two people experience an identical sequence of events to which the concept is applied [Novak]. A concept map is a graphical representation of the concepts of a body of knowledge along with their interrelationships in the form of a directed graph. The map itself consists of a focus question to focus the scope of the map, the concepts are represented as nodes and the relationship between the concepts are represented as directed arcs with connecting words. Figure 3 shows a concept map that illustrates the body of knowledge related to concept mapping. The figure illustrates one important aspect of concept maps which is their ability to show large amounts of information in a compact format. The information shown in Figure 3 would have taken several paragraphs of text to explain but it can easily be shown in one concept map. To highlight more details of the concept map, the combination of a concept, connecting words, and target concept must form a proposition. In the case of concept maps a proposition takes the form of a statement that is either true or false. The requirement for propositions to be either true or false in turn requires that concepts in the map be defined otherwise the propositions become nonsensical. For example, the following are propositions taken from Figure 3, “Concept Maps represent Organized Knowledge” and “Focus Question(s) are Context Dependent”.

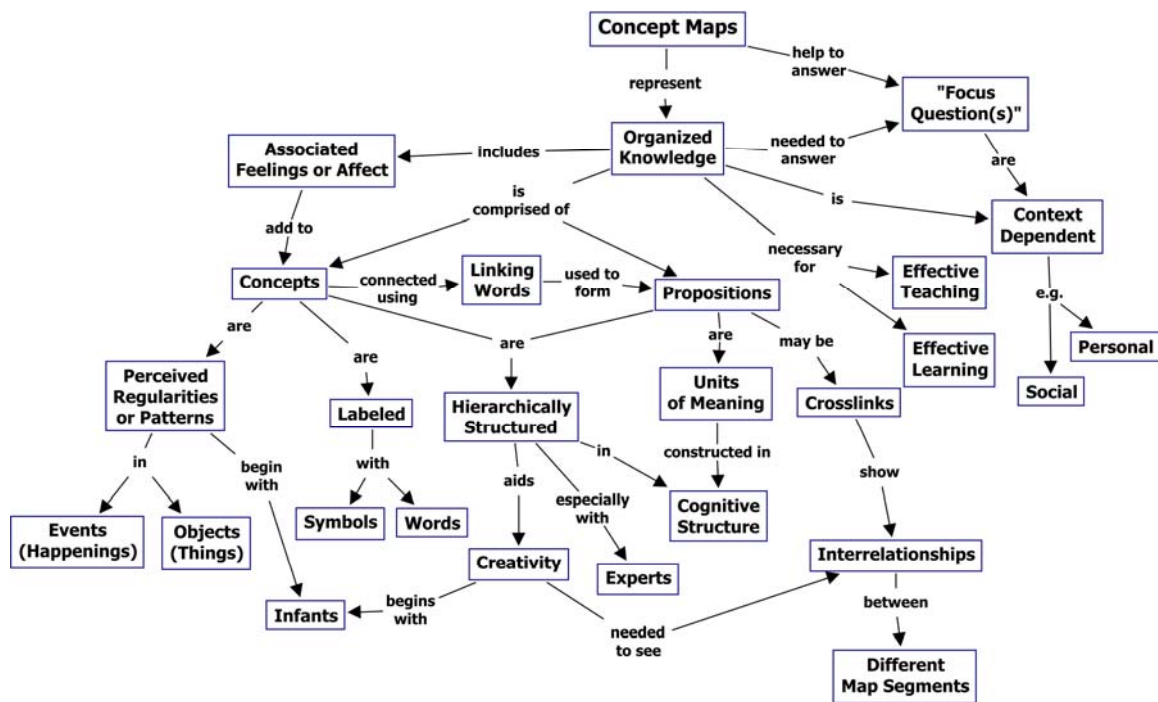


Figure 3 Concept Map Example [Novak 1998]

Concept maps were developed by Professor Joseph D. Novak of Cornell University in the 1970s based on David Ausubel's theory of meaningful learning [Ausubel][Novak 1998]. Concept maps initially came out of research Professor Novak and his research team was conducting into how to best determine what children understood about a subject both before and after instruction. This information is important to the field of education as a means to evaluate teaching techniques. Since their inception, concept maps have been used in several different fields and contexts due to their ability to be used as communications tools that can graphically represent complex and intricate content. They are often used in cooperative environments to construct knowledge i.e. brainstorming. As teaching tools, concept maps can be used to create lessons by identifying key concepts and the interrelationships. When used to present the outline of a domain, concept maps become an organizer and after learning they become integrative tools [Novak]. Finally, since concept maps combine both conceptual and graphical aspects the theory of dual coding predicts that when the same conceptual information is presented in a concept map format then it will lead to better memorization of

the material [Bruillard]. Novak [Novak 1998] outlines a procedure for creating concept maps which was used in the creation of the map presented in this work.

Of particular relevance to this work is the concept map's ability to be used in collaborative environments by users with differing backgrounds and the ease at which the technique of concept mapping can be applied.

PROPOSED SOLUTION

As illustrated in previous sections, the two overarching problems in software engineering that this work addresses are the lack of a common conceptual model that supports a shared understanding of a system and the difficulty in mapping between views or conceptual models of a system.

Any solution to the problems presented above have certain requirements that must be met. Due to the complexity of software systems the shared understanding must convey large amounts of information efficiently and effectively in a simple manner. This requirement will also address the problems of large documents becoming impenetrable to new users and the user's reluctance to update documents as required [Smolander]. Due to the number of problem domains that software engineering is applied to any solution must be very expressive and must be able to support many problem domains as well. In order to support the broadest set of stakeholders possible the solution selected must be easy to learn with little or no formal training required. The solution should support a collaborative environment so that issues regarding subjective and ad-hoc translations can be avoided by performing the work in a group environment. Finally, the shared understanding should not be limited to one phase of the software development life cycle but be incorporated as a fundamental part of the process.

Based on the problems presented and the requirements placed on the solution the solution selected in this work is to provide a software engineering artifact based on the techniques of concept mapping. The concept map will be used to provide this shared understanding of the system by graphically illustrating the concepts identified in the requirements document with associated behavior and restrictions. The requirements document will define the concepts of the system as well as illustrate relationships between the concepts in the system. The definition of the concepts is necessary to provide primitives for the concept map. This concept map will live throughout the lifecycle process and provide a common conceptual model of the system

context, manage the volume of information and enable a common language for referencing components of the system².

With a common terminology and understanding of the concepts of the system the conceptual model translations would no longer be necessary at the highest level. A common model of the system will exist that will provide stakeholders a starting place to begin discussions about the system thereby creating a shared understanding of the system. Since the model translations would not be required, because a general purpose model exists in its place, the model translations would not be subjective nor ad-hoc but defined early on at the requirements stage thereby removing any inaccuracies. Next, since the concept maps are living documents based on requirements documents they will not be out of date because of the centralized nature of the map and lack of conceptual changes in a system. Additionally, concept maps are effective tools for displaying large amounts of information in a compact format and lend themselves to being created in a collaborative environment [Kremer]. As described in Novak, concept maps have been used in many different problem domains so they address the issue of being robust and applicable to many domains. Another advantages of concept maps is the ease at which the map can be visualized and understood [Novak][Turns][Darmofal][Navarro]. Finally, concept maps are often taught to elementary school students so they are easily learned with little formal training being required.

² This process will be referred to in this thesis as *Concept Map Based Software Engineering*.

METHODOLOGY

In order to explore the use of concept map based software engineering, the concept mapping technique was integrated into the development process of the senior/graduate level COMP-190 Software Engineering Project course. A document describing the concept map based software engineering method was created which included a tutorial on the creation on concept maps (see Appendix A). This tutorial was presented to the COMP-190 class on February 6, 2007 for approximately 45 minutes. The portion of the presentation dedicated to the creation of concept maps lasted approximately 20 minutes. Feedback from the class during the concept map training indicated that the students would like a guide for defining relationships between concepts. While such a fixed vocabulary is not part of the procedure for creating concept maps as outlined by Novak [Novak 1998] such a guide has been created as part of this work. The guide provides a mapping between commonly encountered UML notations representing object oriented relationships and concept map connecting words. The guide itself will be presented in the results section. Neither the tutorial materials nor the PowerPoint® presentation were made available after the presentation which was purely an oversight but it proven serendipitous as will be seen in the conclusions. Based on this brief introduction two of the four software teams independently decided to incorporate concept maps into their process.

In order to evaluate the use of the concept map technique notes were taken during the concept-map and non-concept map presentations to gauge the level of understanding by the other teams, the customer, and the professor. One of the key things that were observed was the type of questions that were being asked. For example, a probing question is indicative of someone who understands the design but is digging deeper. This would indicate that the presenters were able to communicate the design well. A superficial question would indicate that the presenters were not able to communicate the design as well as the group in the previous example. The primary evaluation criteria that was used was how quickly and easily the customer and non-concept map teams can understand the concept map design presentations compared to the other presentations.

RESULTS

This section consists of three sections, *Study Results*, *Suggested Partial Mapping from UML to Concept Map Connecting Words* and *Conceptual Model Mapping Results*. *Study Results* will discuss the experiences of the concept map technique in the COMP-190 class. The second section, *Suggested Partial Mapping from UML to Concept Map Connecting Words*, will describe a the mapping between UML notation and concept maps that was developed as a result of the interactions with COMP-190. Finally, *Conceptual Model Mapping Results* will illustrate a technique for mapping between different views of a system using a concept map as a common conceptual model.

Study Results

The key aspect of the concept map based software engineering process is that concept maps be used throughout the software engineering process. One advantage of using the concept map is to bring stakeholders up to speed quickly and facilitate communication with the customer during meetings by mapping the system architecture to the concept map. In addition, based on previous research, the requirements elicitation should be more productive than a non-concept map technique. This would enable the concept map-based technique to result in a better product due to fewer miscommunications [Freeman][Kop]. Finally, the consistency between presentations and the linkages to the design, requirements, and documentation should ease understanding of the system for the customer, users and maintainers of the system.

There are three presentations defined as deliverables for the class, an overview presentation, a design presentation and final presentation. Unfortunately, by the time the class received training on concept maps the project teams had already done initial rounds of client interviews so the effectiveness of creating and using the concept map during user interviews in the requirements elicitation phase can not be evaluated in this pilot study. However, other work has shown that the use of concept maps in requirements elicitation leads to more well defined requirements. An example of using concept maps in requirements gathering is shown in the work done by the Centers for Disease Control to create a logic model for the Preventions Research Center's Program [Anderson] and in [Freeman][Kop]. The Preventions Research

Center's Program is a nationwide network of researchers, public health agencies and community groups that conduct applied research in disease prevention and control. Concept maps were used to help create the *Framework for Program Evaluation in Public Health* which is a tool used to evaluate program improvements processes and public health decision making [Anderson]. The CDC created two concept maps to elicit information from the academic researchers, public health agencies and community groups, a total of 145 national stakeholders and 135 local stakeholders gave input to the creation to the national and local concept maps [Anderson]. The CDC sees the use of concept maps as providing a valuable source of data by establishing a common conceptual model of a very large program. The logic model that was generated using the concept maps is meaningful for stakeholders as it incorporates input from program partners and establishes program expectations [Anderson]. The work at the CDC indicates concept maps are valuable tools for gathering user requirements from a large geographically disbursed set of stakeholders.

Based on the concept map training given to the class, two out of four teams, Teams 3 and 4, decided to incorporate concept maps into their development process. However, upon closer inspection of Team 1's UML component diagram it appears that the component diagram does not comply with UML 1.4 nor UML 2.0 notation. Instead, it appears that Team 1 merged portions of the UML component diagram notation with the concept map technique of using connecting words to describe the relationship between concepts. In other words Team 1 made a UML component diagram into a diagram that more closely resembling a concept map.

The two other teams that did decide to use 'pure' concept maps, Teams 3 and 4, did create concept maps for the design presentations. One of the risks of introducing any new technique is that if the teams were required to generate concept maps then the concept map would not be seen as a tool for communication but as extra work and the implementation of the map would be approached half heartedly. This risk did not materialize since the teams were not required to use the concept mapping technique. A positive point is that the concept map, which was not required and which required more work than required by the course were completed. This indicates that the maps were seen as useful artifacts and worth the effort. In

addition, the maps were used throughout the development process again indicating that there was utility in their creation.

Team 3

Team 3's task is to design an Online Collaborative Lesson Planner. The lesson planner is used as part of a course for K-8 teachers and sponsored by the Technical Education Research Centers a non-profit education research and development organization dedicated to improving teaching and learning of mathematics, science, and technology. The Collaborative Lesson Planner helps to train student-teachers to become teachers. The team developed two concept maps which are reproduced in Figure 4 and Figure 5.

Analyzing the concept maps they identify key concepts of the system and their relationships. However, since the concept map training was received late in the software engineering process the creation of the concept map was done during the design phase. Therefore, certain design concepts such as "Database Server" and "App Server" are in the map which may be superfluous to the concepts of the system. In addition to the two concept maps Team 3 presented their architecture using a "boxes and lines" diagram (Figure 6), a UML Class Diagram and two UML Sequence Diagrams.

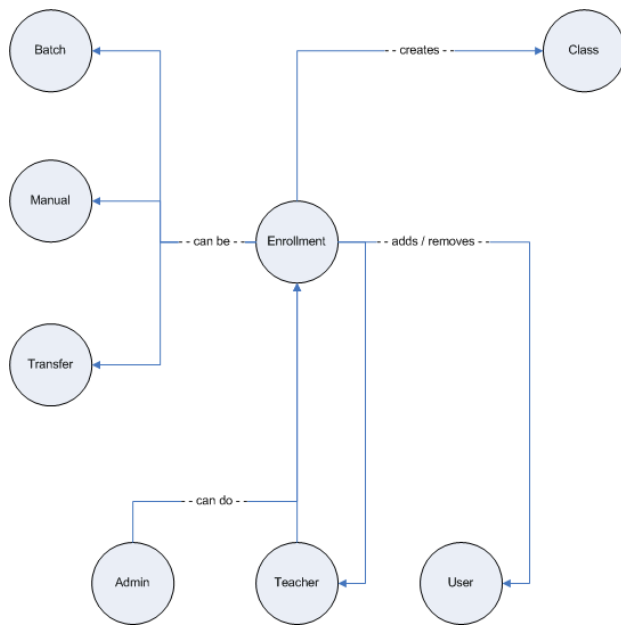


Figure 4 Team 3 Concept Map Showing Enrollment

When Team 3 presented the system architecture, the second deliverable to include the concept map of the system, the class consisted of all members of COMP190 and two industry professionals from Avaya who had no previous exposure to the project. The presenter walked through the concept maps calling out all concepts and propositions in the system. The concept maps of the system were explained without any issues or questions and all stakeholders were brought back up to speed on the Online Collaborative Lesson Planner. Regarding the concept map, the presenter commented that the entities in the system have very complex relationships and that the roles the entities have can be easily defined with a concept map.

When the presenter began explaining the “boxes and lines” diagram shown in Figure 6, which describes the physical layout of the web pages of the system there were several questions. Specifically, the industry professionals were confused by the notation of the diagram asking if there was any significance to the shape of the boxes. This began a lengthy discussion with the entire class about what the diagram was meant to represent. For example, what was the significance of a cgi script in an oval versus a cgi represented by a triangle.

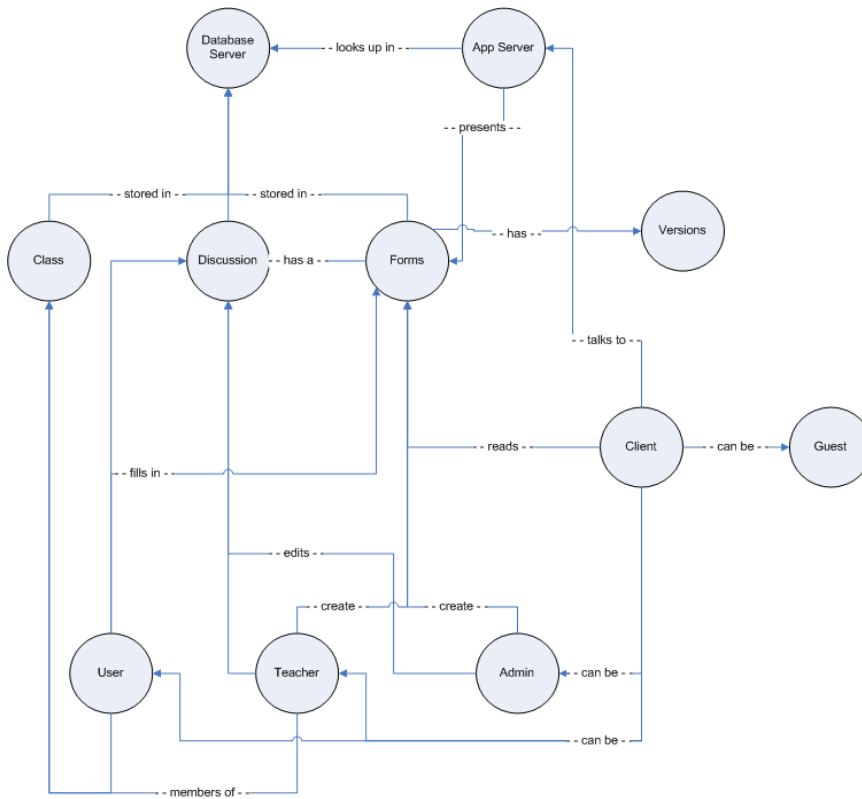


Figure 5 Team 3 Concept Map Showing Discussions and Forms

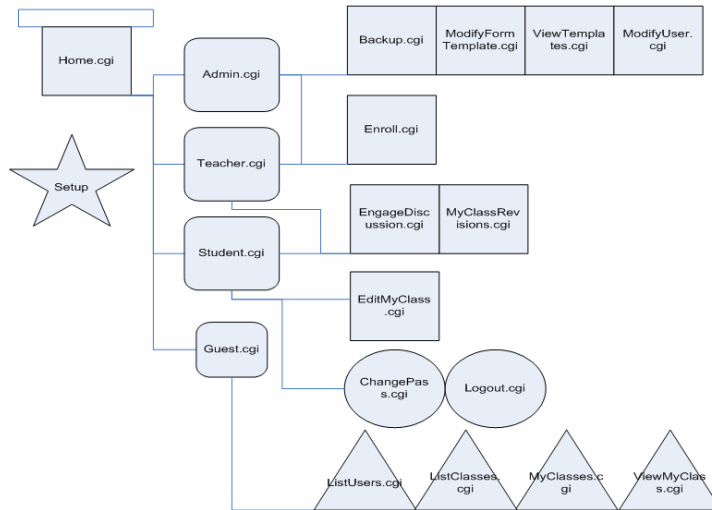


Figure 6 Team 3 "Boxes and lines" diagram

This shows that the concept map's notation and explanations were clear since even stakeholders that had previously not been exposed to concept maps, their notation, nor the system they represented came up to speed on the Online Collaborative Lesson Planner system quickly. This is evident because when the "boxes and lines" diagram was presented the stakeholders were familiar with the system and were able to ask probing questions about the diagram and how the "boxes and lines" diagram fit in to the overall system.

Team 4

Team 4 is tasked with creating a system to provide real-time tagging of phone conversations using an instant messenger interface. The system will allow entire conversations to be tagged, points in conversations to be tagged and audio segments of conversations to be tagged. The project is sponsored by Avaya a company providing communication network design, building, and management. The concept map team 4 created is reproduced in Figure 7.

Team 1

Team 1 was tasked with creating widgets to perform artifact and image searches of the Perseus Digital Library. Widgets are small user interface components that can be added to computing devices for specialized functions. While Team 1 did not formally use concept maps, as noted above, they did create a UML component diagram that did not conform to UML 1.4 nor UML 2.0 notation. Team 1's 'UML based concept map' is presented in Figure 8 (top) along with a UML component diagram using standard notation (bottom).

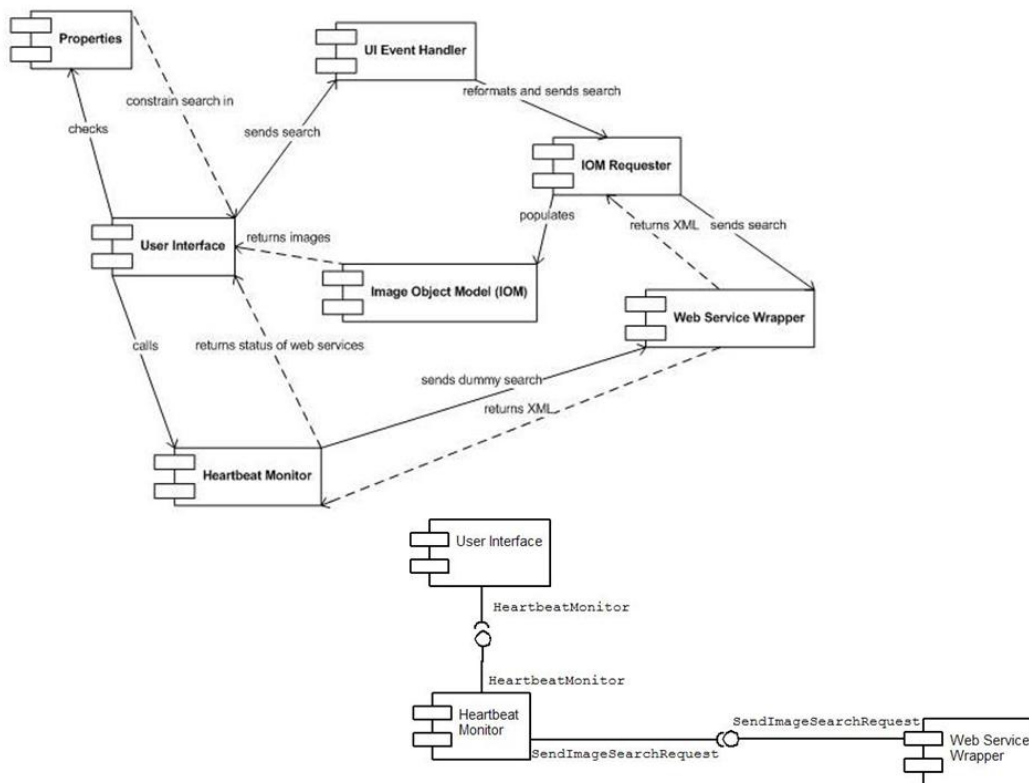


Figure 8 Team 1's UML Based Concept Map

Examining Team 1's UML component diagram on the top half of Figure 8 certain properties stand out. Specifically, the component document is not standard UML notation and the edges connecting the components are labeled with connecting words. For example the "Heartbeat Monitor 'sends dummy search' [to the] Web Service Wrapper". This notation is more

indicative of a concept map where the components represent concepts and the relationships are shown using connecting words. A standard UML component diagram is presented in the bottom half of Figure 8. The standard UML notation is quite different with only interface names being visible and the 'lollipop' connectors being used to show the association between components. Team 1 therefore used the idea behind the concept map, however, they employed it in an unproven notation.

Teams 1 and 2

Finally, teams 1 and 2 presented their designs to the class. For the most part these presentations consisted of UML diagrams of the systems. Of particular interest is that the presentations for teams 1 and 2 took considerably longer than those for teams 3 and 4 due to the number of questions they received from the class regarding the purpose of the systems. It is not clear if the questions resulted from the lack of understanding of the system due to the use of UML, presentation style or the class' level of concentration by the time teams 1 and 2 presented.

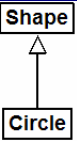
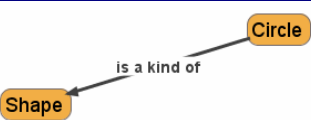
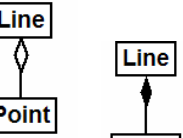
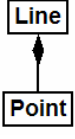
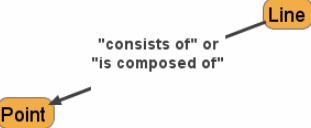
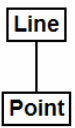
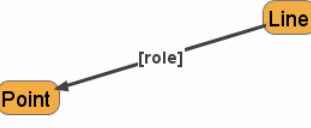
Suggested Partial Mapping from UML to Concept Map Connecting Words

This section will provide a guide for a suggested vocabulary for mapping UML diagrams to concept map connecting words. As mentioned in previous sections, this guide came from observations that users of concept maps that are also users of UML required some basis for linking the two notations in a structured manner. This suggested mapping is done with a strict adherence to requiring that concepts be defined and requiring that all propositions be either true or false. Using these rules similarities to a formal system can be seen. In basic terms a formal system consists of concepts and assertions about their properties [Preparata]. Ideally the absolute nature of both the definitions of the concepts and the validity of the assertions could be determined. However, this goal can not be achieved since defining concepts requires the use of other concepts which in turn requires the use of even more concepts. This cycle continues endlessly. In order to break the cycle a method similar to that used in constructing a formal or mathematical systems is needed. In the construction of a formal system a set of concepts based on a context are selected. These concepts are called primitives and are used

without being defined. Now, any new concept added to the system can not be added without first explaining it using the primitives and other concepts previously defined in the system [Preparata]. Since the propositions can not be readily evaluated without human interpretation coupled with cross references to perhaps multiple other artifacts the concept map is more similar in nature to propositional logic [Peters]. However, the concept map is superior to natural language prose since it consists of statements that can be evaluated independently rather than statements intermixed with connecting sentences for the purposes of making the prose readable.

This section will present a table illustrating UML notations and the suggested proposition language to be used in the associated concept map. Currently the mapping between concept map connecting words and a subset of the full set of UML notations are defined. It is envisioned that future work can build on this mapping and complete the mapping for the remaining UML notation.

Table 1 Partial Mapping of UML Notation to Concept Map Connecting Words

UML Notation	Concept Map Proposition Text
 <p>Generalization</p>	 <p>is a kind of</p>
 <p>Aggregation</p>  <p>Composition</p>	 <p>"consists of" or "is composed of"</p>
 <p>Association</p>	 <p>[role]</p>

UML Notation	Concept Map Proposition Text
<p>Multiplicity</p>	<p>has 1 or more</p>
<p>Multiplicity</p>	<p>may have</p>
<p>Multiplicity</p>	<p>has many</p>
<p>Multiplicity</p>	<p>has any number other than 2 or 5</p>

Conceptual Model Mapping Results

In addition to providing a shared understanding of a system a concept map can also be used for other uses in the software development process. One of these uses was uncovered during thesis preparation meetings with Dr. Stafford and is the ability to use the concept map to map between different views of the system. Initially the concept map was seen as a communications tool to map from different views of the system to a common conceptual model to provide a shared understanding. Extending this idea to the next step became that if views ‘A’ and ‘B’ could be mapped to a concept map ‘C’ then ‘C’ could be used to bridge between the views ‘A’ and ‘B’. This idea proved viable and was added to the scope of the thesis due to the importance of the problem it addressed. This idea has been explored and the preliminary observations have submitted to the Third International Conference on the Quality of

Software-Architectures in a paper entitled “Using Concept Maps to Enhance View Navigation”.

Since the idea of conceptual model mapping did not materialize until late in the project the teams were not presented with the idea. To illustrate the mapping of conceptual models, Team 3 class diagrams, concept maps and use case diagrams will be mapped. A use case to class diagram mapping was selected because often times the association of classes in a system involved in the implementation of a use case are not obvious. This mapping example will attempt to make the association clear. A portion of Team 3’s use case diagram is presented in Figure 9. A random use case was selected from the diagram, Teacher <<uses>> Enrollment, Admin <<uses>> Enrollment.

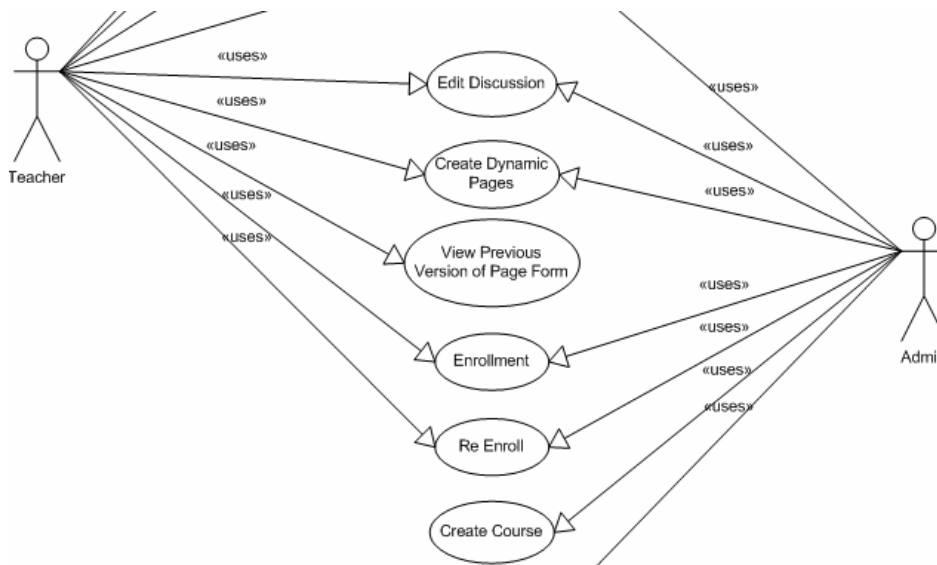


Figure 9 Partial view of Team 3’s Use Case Diagram

The concept maps that Team 3 created were shown in Figures 3 and 4. In order to illustrate all the relevant concepts in the Enrollment use case the two concept maps had to be merged, the author performed the merge and the resulting concept map is presented in Figure 10.

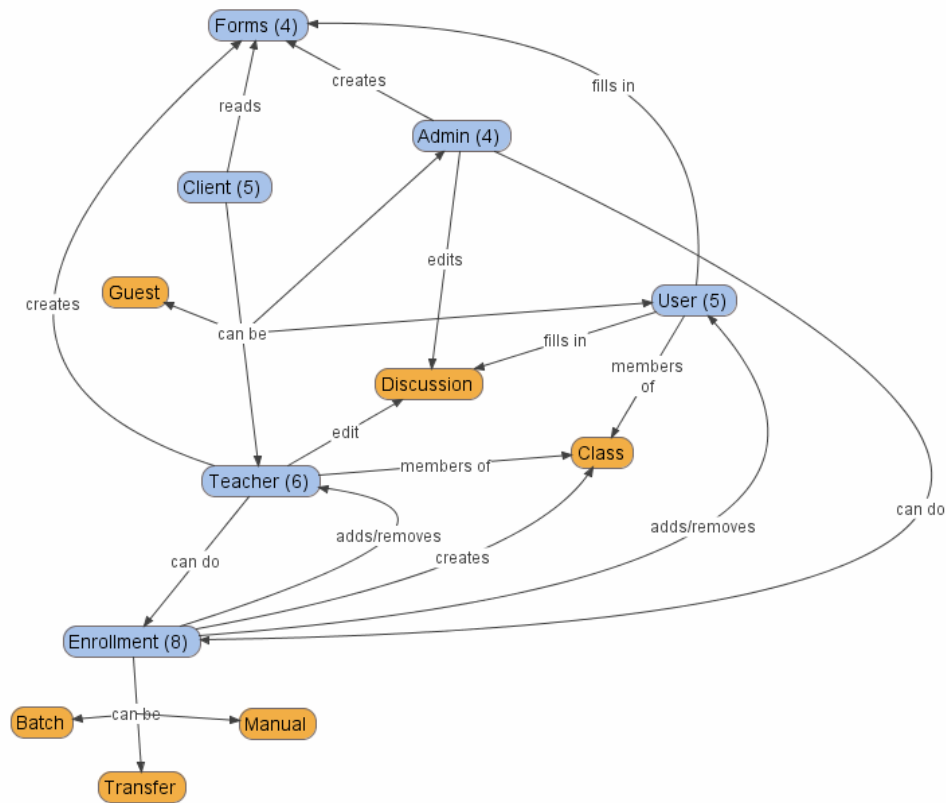


Figure 10 Team 3 Concept Maps Merged Together

All relationships and concepts are reproduced in the merged map. The only omissions are the Database Server and Application Server since they are more of a design detail that a system concept and are also not relevant to the Enrollment use case. Two details were added to the map, those are the addition of the coloring of the Enrollment, Teacher, User, Client, Admin and Forms concepts to highlight them as key concepts in the system. Those concepts were selected as key concepts based on the number of relationships that they participate in. The rationale being that if a concept participates in several relationships then it is important to the system and therefore is a key concept to the system. This brings us to the second addition to the map, the number in parenthesis for each of the blue-colored concepts. The number represents the degree of the node, in other words, the total number of arcs entering and leaving the node. Since the arcs in a concept map represent relationships it would appear that the higher the degree of a node the more relationships that node participates in. Returning to

the merged concept map, based on the number of relationships the concepts Enrollment, Teacher, User, Client, Admin and Forms are ranked in the order of importance. The relevance of the ranking is that during mapping, any part of the implementation that maps to key concepts should be examined for scalability and performance issues since those parts of the implementation will become key parts of the implementation as well. The degree of the node was selected rather than giving higher value to arcs entering the node or to arcs leaving nodes in order to compensate for the order that concept map was created. To illustrate, in Figure 10 the concept Discussion has degree of three. An equally valid concept map for Discussion could be created by reversing the arcs coming into the node and changing the connecting words. For example currently Discussion has three arcs entering the node and participates in the following relationships, “Teacher edits Discussion”, “Admin edits Discussion” and “User fills in Discussion”. However, these relationships could be changed to be “Discussion can be edited by Teacher”, “Discussion can be edited by Admin” and “Discussion is filled in by User”. This modification would result in three arcs leaving the node however, it would not change the degree of the node and thus not change the importance of the concept to the system. This modification also illustrates another important aspect which is how to represent multiplicity. All the concepts in the map are singular yet in practice there may be many instances of the concept in the system. For example, “Discussion can be edited by Teacher”, does this mean that there is a one to one mapping such that a Teacher can only edit one discussion? Does the relationship mean a one to many situation where one Discussion can be modified by many Teachers? Finally, is this a many to many relationship where many Discussions can be modified by many Teachers? The topic of multiplicity is further discussed in Appendix A in the mapping of UML to concept maps. The solution to this ambiguity is to be as precise as possible in the connecting words used in the map which includes adding additional relationships that call out how many concepts are involved in the relationship.

Mapping between Team 3’s use case diagram, merged concept map and class diagram proved an easy exercise. What proved difficult was the visualization of the mapping for the three conceptual models. The mapping is illustrated in Figure 11. Initially arrows were drawn between concepts, use case elements and classes however, the lines proved to be too cumbersome due to the amount of overlapping. Color was added to highlight the concepts

involved in the mapping which helped navigate the diagrams however the linkages were not obvious. The best technique found thus far has been a combination of colors highlighting the concepts, use cases and classes involved in the mapping along with a table describing the relationships between the conceptual models explicitly. This table was required because in some cases parts of a class, for example a method, were needed in one part of the mapping and parts of the same class were needed for other parts of the mapping. The table for Team 3's mapping is presented in Table 2.

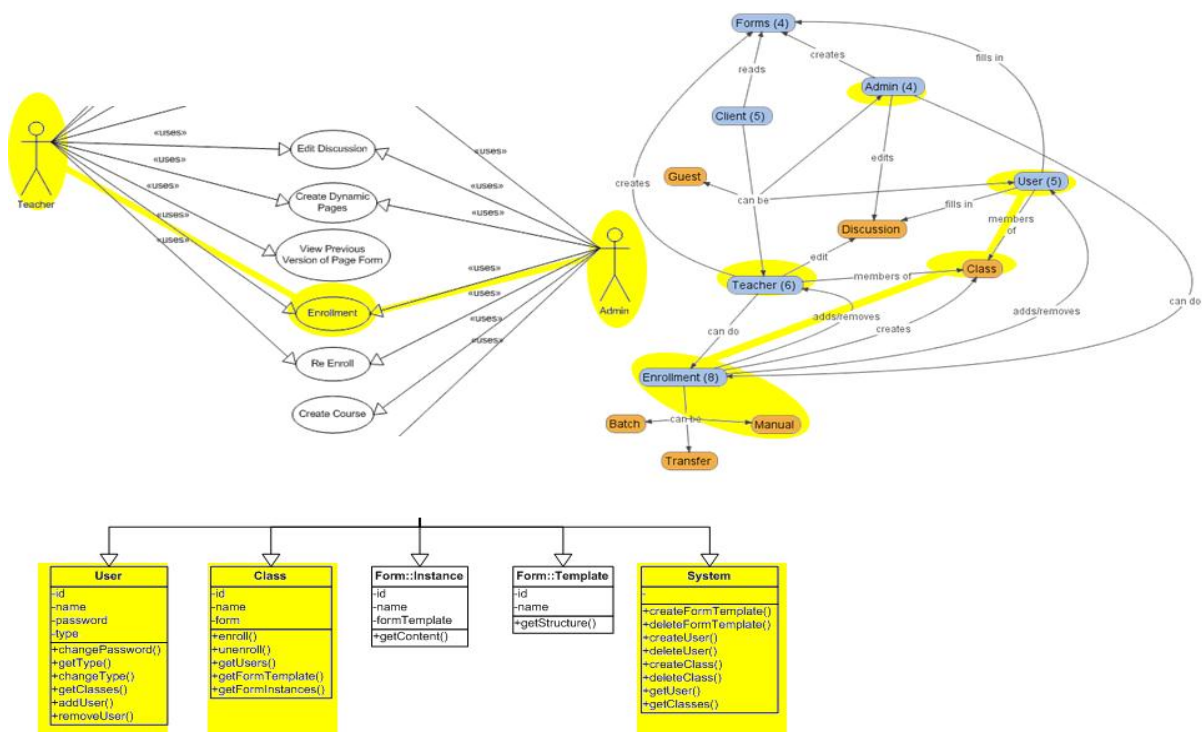


Figure 11 Mapping Team 3's Models

Table 2 Model Mapping

Use Case	Concept Map	Class Diagram
Teacher	Teacher	Instance of the User class
Admin	Admin	Instance of the User class

Enrollment	Enrollment “Can be” Manual	System::createClass
	Enrollment “adds” User	System::createUser, returns User,
	Enrollment “creates” Class	Class:enroll()
	Enrollment “adds” Teacher	

The model mapping table consists of three columns, Use Case, Concept Map, and Class Diagram. The Use Case column uses the vocabulary from the Use Case diagram, in this case Teacher. The Concept Map column uses the vocabulary from concept map and illustrates the mapping from the Use Case to the Concept Map. The third column, Class Diagram, uses the terminology from the class diagram and performs the mapping between the Concept Map and the Class Diagram. In this case Teacher maps to an instance of the User class.

Going further into the mapping, the Enrollment use case maps to the Enrollment concept in the concept map. The direct mapping between the Enrollment use case and the concept map is that “Enrollment ‘Can be’ Manual’ since in the case of the Admin or Teacher they are performing manual enrollment. However, the Enrollment concept has several other relationships associated with it. Specifically, “Enrollment ‘adds’ User”, “Enrollment ‘creates’ Class” and “Enrollment ‘adds’ Teacher”. Some of these activities are significant to the class diagram and are pointed out in column three. The “Enrollment ‘creates’ Class” relationship maps to [System::createClass], “Enrollment ‘adds’ User” maps to [System::createUser, returns User], “Enrollment ‘adds’ Teacher” maps to [Class::enroll()] and finally the created user is enrolled using [Class:enroll()].

Even with the difficulty in visualizing the conceptual model mapping the mapping technique proved useful for identifying relationships between the conceptual models of the system. For example, one issue with Team 3’s system is that the concept of User is overloaded. User appears in the concept map as a subordinate to Client but in the class diagrams the class User appears to take on a more significant role as a base class for all other user types such as Teacher, Admin, et cetera. This inconsistency should be addressed so that a clear conceptual model of the system is presented. Another interesting result from the concept map conceptual model mapping is that the Enrollment concept has several relationships in the concept map

and therefore is considered an importance concept. Yet it does not appear explicitly in the class diagram of the system. As pointed out in the mapping presented in Figure 11, enrollment is hidden in the implementation of the system and is spread across several classes, System, User and Class. Since such a critical piece of the system is spread across many different classes it will become difficult for maintenance programmers to debug problems in the Enrollment feature as well as add new capabilities to it.

CONCLUSION

A key goal of the thesis was to enable a shared understanding between various stakeholders using concept maps for software engineering. Evidence that this goal was obtained can be seen in the presentation Team 3 gave. Team 3 was able to bring several stakeholder groups, outside development groups in this case, up to speed on their problem and solution using concept maps as the key discussion driver. There were no issues or questions when Team 3 presented their problem to the class using the map however when the “boxes and lines” diagram representing a web site deployment was presented several questions and issues were brought forward. This indicates that there was not a problem with the presenters style but with the content of the presentation when moving from the concept map to the “boxes and lines” diagram. In addition, when UML sequence diagrams were presented there were additional issues and questions. The questions however were not context related but were related to implementations of the classes themselves. This indicated that the class had a shared understanding of where in the system the sequence diagram was applicable but had issues with mechanics of the sequence diagram itself.

As mentioned previously some of the requirements of any proposed solution are the abilities to present large volumes of information in a compact format and to support many problem domains. Clearly based on the work of Team 3 and 4 these two requirements were met. Both teams could represent their entire architecture in one concept map and since both teams were working in very different problem domains it is clear that many problem domains can be supported. Other requirements of the solution are that it be easy to learn and it be able to be done in a collaborative environment. Constructing the artifact in a collaborative environment has been shown as a way to make better designs. It is unclear from this experience if a better design resulted from the use of concept maps or not. However, concept maps are effective communication tools as seen in all concept map based COMP-190 group presentations and as such they have been shown to help communicate between different user groups. This communication will enable the better exchange of ideas and concepts thus resulting in a more complete design. Regarding the ease of learning, the class was given a 20 minute introduction

to concept map creation and two teams were able to create the maps and make effective use of them. In addition, as noted above, the material from the presentations was not left for the class so the 20 minute introduction was all the training the class received. Therefore, training is not an issue.

Mapping between views proved to be the most difficult aspect of this work. Specifically the issues of visualizing between the conceptual models was the most difficult. In the end a combination of color highlighting the different parts of the system being mapped and a table specifying what parts of the conceptual models were being mapped proved to be the best solution. It is anticipated that this solution too will suffer from scaling issues when the conceptual models being mapped get too large. Therefore, more work visualizing the actual mapping with more complex systems is needed. Also, a mapping using a more complete set of UML conceptual models, Connector and Component views, and Department of Defense Architectural Framework (DoDAF) conceptual model examples is needed particularly with large systems.

Additionally, a method to evaluate the effectiveness of using concept maps in software engineering should be researched and developed. One technique that may prove valid is active design reviews [Parnas]. In active design reviews a set of reviewers are given questionnaires and specific objectives. Using this technique reviewers could be asked to locate requirements in both concept maps and requirements documents and locate design details in both concept maps and system views. This would potentially enable the robustness and easy of use of the concept map to be evaluated.

Finally, it is envisioned that future work can build on the UML notation to concept map connecting word mapping. This mapping is needed to leverage the extensive work that has gone into the definition of UML and use the ideas developed there in concept maps.

The results from this study were very encouraging as to the effectiveness of mapping between conceptual models using a concept map as a bridge and using a concept map to provide a shared understanding to different stakeholder groups. Clearly more work is required in more controlled circumstances but this work proved quite successful.

APPENDIX A CONCEPT MAP BASED SOFTWARE ENGINEERING
TUTORIAL

DEPARTMENT OF COMPUTER SCIENCE
TUFTS UNIVERSITY

INTRODUCTION TO THE CONCEPT MAP BASED SOFTWARE
ENGINEERING METHODOLOGY

©2007 THOMAS HUBBARD

Table of Contents

Appendix B Concept Map Based Software Engineering Tutorial	36
Introduction	39
Methodology	42
Conclusion.....	50
Bibliography.....	51

TABLE OF FIGURES

Figure 15 View translation from UML to a Customer-friendly view.	40
Figure 16 UML Class Diagram	43
Figure 17 Example of the Noun Extraction Technique	45
Figure 18 First version of the concept map.	47
Figure 19 Step 7: Adding lines and linking words	47
Figure 20 Step 8, Iteration 1.	48
Figure 21 Final iteration of the concept map.....	49
Figure 22 Comparison of UML to Concept Maps	50

Introduction

The development of software systems is commonly done in a series of logical phases or steps. Each phase in this process produces an output that serves as a deliverable product for that phase and as an input to the following phase(s). Each phase of the process serves a different purpose and the form the output takes for each phase is selected based on the best choice for the job at hand. For example, a requirements phase may output a natural language document describing the system and a design phase may output a set of UML documents.

In practice many different people have an interest in the system being developed and take part in many if not all phases of the software development process. This collection of people are known as *stakeholders* in the system[Bass]. The stakeholders, e.g. end-users, program managers, members of other development programs and system designers, all have different professional backgrounds and expertise they will use to conceptualize or view the system. For example, a system architect may conceptualize the system as a set of object-oriented classes and processes whereas a program manager may view the system as a timeline with resources and deliverables. Generally, a stakeholder group will use a particular tool or method for representing their view of the system. For example, the development team may use UML diagrams to visualize the system whereas a program manager may use a GANTT chart.

When members of several different stakeholder groups meet a problem arises, the different conceptual views of the system must be translated from the conceptual view used by one group to a more general view for use with a broader audience. In this thesis, the activity of translating one stakeholder's view to another is known as *view translation*. An example of a view translation is shown in Figure 12. Consider two stakeholders that are meeting to determine if an architecture will meet all the requirements imposed on the system. One stakeholder is the system architect and the other is the customer. The system architect is using UML to represent the system so she must translate the view into something the customer understands.

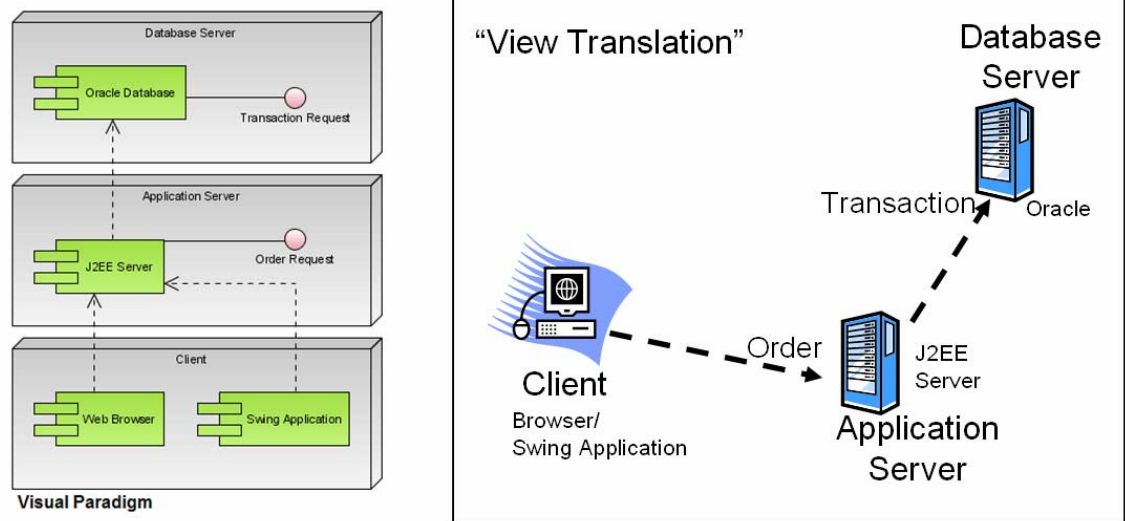


Figure 12 View translation from UML to a Customer-friendly view.

Referring to Figure 12, the image on the left represents a three tier client server system in UML using packages. The image on the right represents what is presented to the customer, typically using a business application such as Microsoft's PowerPoint®.

The problem with view translations are that they are 1) inaccurate due to the subjective nature of the translation 2) inconsistent since the translations are done in an ad-hoc manner and therefore vary from presenter to presenter 3) time consuming to produce 4) inaccurate when translated from a more robust representation of the system and 5) out of date with reality since work must be done to keep the translated view in line with the 'stakeholder' view. All of these problems result in miscommunications and wasted time and effort.

To combat the deficiencies presented above, this work proposes a software engineering artifact known as a concept map be used to graphically illustrate the concepts identified in the requirements document with associated behavior and restrictions. This concept map will live throughout the lifecycle process and provide a common view of the system context, manage the volume of information and enable a common language for referencing components of the system. Rather than simply being an output of one phase of the software development

process, the concept map would span multiple phases and provide a context from which stakeholders can derive a common terminology and understanding of the system³.

With a common terminology and understanding of the concepts of the system the view translations would no longer be necessary at the highest level. A common view of the system will exist that will provide stakeholders a starting place to begin discussions about the system.

The advantages of the concept map approach are that view translations would not be subjective nor ad-hoc but defined early on at the requirements stage thereby removing any inaccuracies. Next, since the concept maps are living documents based on requirements documents they will not be out of date because of the centralized nature of the map and lack of concept changes in a system. Time will be saved preparing for meetings since the common view of the system is complete and everyone is familiar with the map. Finally, if a stakeholder is not familiar with the map, one of the advantages of concept maps is the ease at which the map can be visualized and understood [Novak][Turns][Darmofal][Navarro].

³ This process will be referred to in this thesis as *Concept Map Based Software Engineering*.

Methodology

The concept map based software engineering paradigm centers around the use of a concept map to provide a context from which stakeholders can derive a common terminology and understanding of the system being developed. The concept map differs from documents used in other paradigms in that the concept map will span multiple phases of the software engineering process.

It should be pointed out that any software engineering methodology can be made 'concept map based' by the integration of a concept map as described above. The steps required to create a concept map for the purposes of concept map based software engineering will be described in the following sections.

The first step in the concept map based software engineering paradigm is to create the concept map. The creation of the concept map is similar in nature to the class modeling activity done in OOA. In class modeling the classes of the system and their attributes are extracted. The extraction of the classes can be performed by analyzing the use cases of the system or by noun extraction [Schach]. The selection of the technique use case analysis or noun extraction depends, primarily on the analyst's familiarity with the domain. In an unfamiliar problem domain noun extraction is often the best fit. The differences between concept map creation and class modeling are mainly in the users of information and the scope of the activity. Class modeling is generally done by developers using specific notations such as UML with a design mind set, see Figure 13 UML Class Diagram.

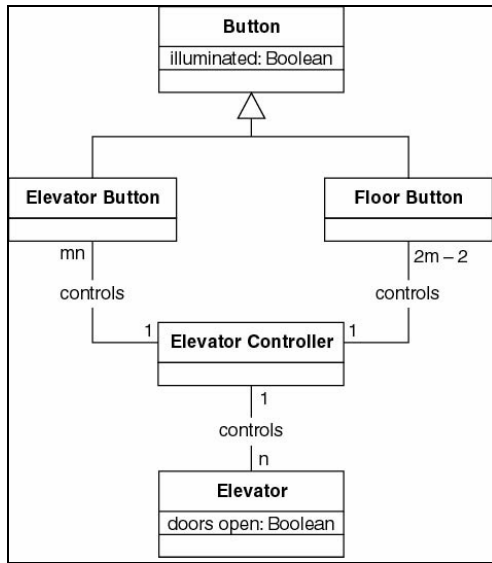


Figure 13 UML Class Diagram

In other words class modeling is domain specific and not easily shared among the many stakeholders of the system. Concept maps were created to aid in describing concepts in an existing body of knowledge and interrelationships between those concepts. The issue at hand is that a system being defined or developed must first have an existing body of knowledge before a concept map can be created. In other words, the problem must be understood.

The following list outlines the steps to create a concept map to be used in the concept map based software engineering paradigm.

1. Create a body of knowledge. In other words the functionality of the system must be understood either through interview, requirements documents, use cases, et cetera.
2. Create a focus question that addresses the domain to be mapped. For example: “What is the XMULE system supposed to do?”. Using this question, identify 10 to 20 concepts that are relevant to this question using the body of information from step 1 and list them. This list is known as a *parking lot* and the items in the list are known as *concept labels* [Novak 1998]. Generally concept labels should be short, e.g. one to three words.
3. Rank the concepts from the most general and inclusive concept at the top to the most narrow at the bottom.

4. Refine the list by adding more concepts if needed, removing concepts, or consolidating concepts that are similar.
5. Begin to build the map by placing the most general concepts at the top.
6. Next, add 2-4 sub-concepts under each general concept on the map. If there are too many sub-concepts under a concept e.g. more than 6 then there is probably a missing intermediate concept that can be used to create new level to the hierarchy.
7. Connect the concepts by directed lines. The lines should contain a few linking words that define the relationship between the concepts.
8. Refine the map by adding, removing, restructuring and consolidating concepts. Since this map will be used for the lifetime of the project it is necessary that the map be as simple as possible. Too much detail will result in a map that will be out of date as the project evolves and too little detail will result in a map that will not meet the map's goal of giving all the stakeholders a common view of the system.

In order to illustrate the process the following section will provide an example of producing a concept map using a simple system description.

Example

Step 1: Understand the Problem

A product is to be installed to control n elevators in a building with m floors. Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause one elevator to move to the corresponding floor. Illumination is canceled when corresponding floor is visited by elevator. Each floor, except the first and the top floor, has 2 buttons, one to request an elevator to move up, up a one or more floors and one to request an elevator to move down one or more floors. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor, then moves in the desired direction. If an elevator has no requests, it remains at its current floor with its doors closed.

Step 2: Discover the Concepts Create a Focus Question

In order for stakeholders to understand what the system is supposed to accomplish or what the purpose of the system is an appropriate focus question may be “*What is the elevator control system supposed to do?*”

Identify Relevant Concepts

Using the focus question, “*What is the elevator control system supposed to do?*”, identify relevant concepts in the system description.

As mentioned above as in class modeling, the techniques of noun extraction or use case analysis can be used to accomplish this step.

A system is to be installed to control n elevators in a building with m floors. Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause one elevator to move to the corresponding floor. Illumination is canceled when corresponding floor is visited by elevator. Each floor, except the first and the top floor, has 2 buttons, one to request an elevator to move up, up a one or more floors and one to request an elevator to move down one or more floors. These buttons illuminate when pressed. The illumination is canceled when an elevator visits the floor, then moves in the desired direction. If an elevator has no requests, it remains at its current floor with its doors closed.

Figure 14 Example of the Noun Extraction Technique

Create a Parking Lot of Concepts

Having performed concept extraction list the concepts in no particular order.

Parking Lot of Concepts (unordered)
Elevator
Building
Floor
Button
Illuminate
First Floor
Top Floor
Current Floor

Step 3: Rank the Concepts

Once the concepts have been identified and extracted they must be ranked from most general or most inclusive to the most specific. For example, the concept of *floor* is more general than the concept of *first floor*.

Parking Lot of Concepts (ranked)
Building
Floor
Elevator
Button
Illuminate
Current Floor
First Floor
Top Floor

Step 4: Refine the Concepts

Refine the list by adding more concepts if needed, removing concepts, or consolidating concepts that are similar. For example, the concepts of floor, first floor, top floor and current floor are related.

Refined List of Concepts (no changes)
Building
Floor
First Floor
Top Floor
Current Floor
Elevator
Button
Illuminate

Steps 5 and 6: Begin to build the map

Begin to build the map by placing the most general concepts at the top. Next, add 2-4 sub-concepts under each general concept on the map. If there are too many sub-concepts under a concept e.g. more than 6 then there is probably a missing intermediate concept that can be used to create new level to the hierarchy.

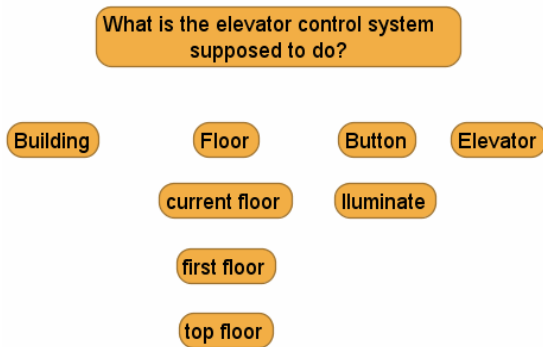


Figure 15 First version of the concept map.

Step 7: Add directed lines and linking words

Connect the concepts by directed lines. The lines should contain a few linking words that define the relationship between the concepts.

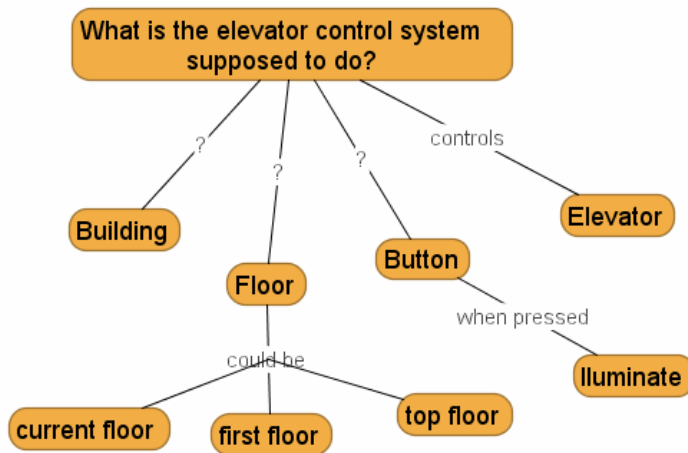


Figure 16 Step 7: Adding lines and linking words

Notice that the relationship between the focus question and the concepts of Building, Floor, and Button are not clear in this step.

Step 8: Refine the map

Refine the map by adding, removing, restructuring and consolidating concepts. Since this map will be used for the lifetime of the project it is necessary that the map be as simple as possible.

Too much detail will result in a map that will be out of date as the project evolves and too little detail will result in a map that will not meet the map's goal of giving all the stakeholders a common view of the system.

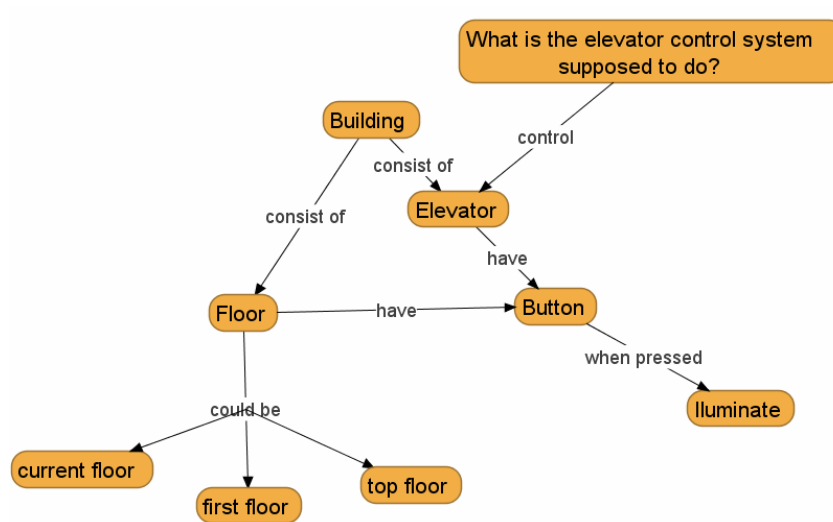


Figure 17 Step 8, Iteration 1.

In Figure 17 the first iteration of the refinement process is shown. What is refined in this iteration is the relationship between the elevator control system and the top level concepts. Now it is clear that the control system controls “elevators” and that “elevators” have “buttons”. Additionally it is explicit that “floors” have “buttons” as well.

The problem with the concept map in Figure 17 is that the concept of “building” has a relationship with both “floors” and “elevators” but not with the elevator control system. This is remedied in the second iteration of the map that follows.

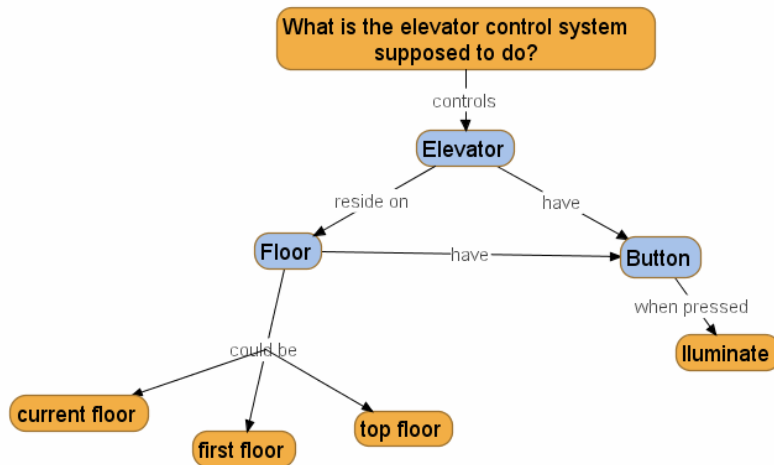
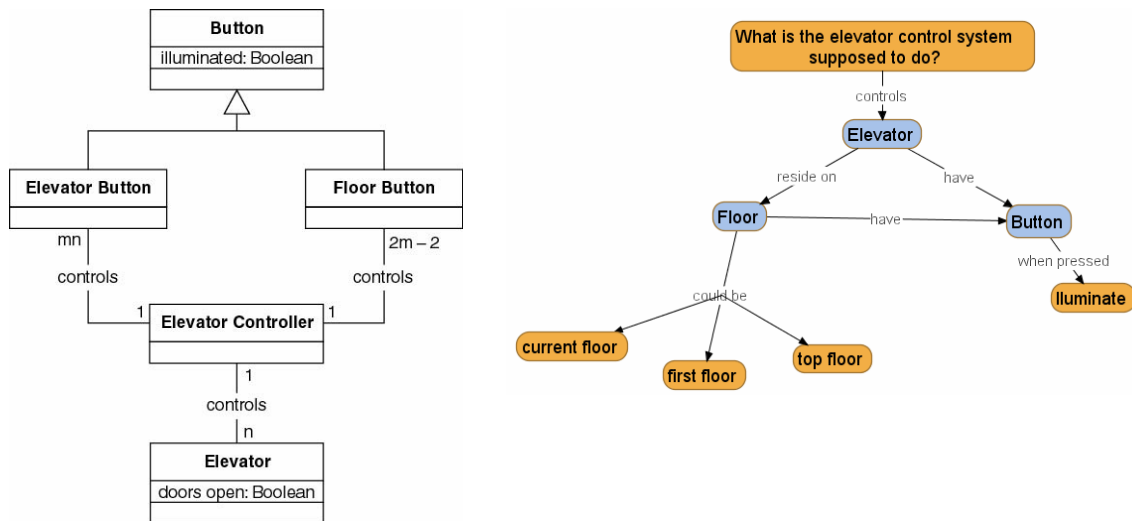


Figure 18 Final iteration of the concept map.

Figure 18 shows the final version of the concept map. In this version the “building” concept has been removed since was determined to be outside the problem domain. All the relationships between the concepts have been defined and the key concepts have been shown in a separate color (blue).

Conclusion

This tutorial has shown an example of developing a concept map for a very simple system. Even though the system shown in the example was simple the stakeholder can grasp the key components of the system much easier than if the design were presented in a more traditional modeling language such as UML (see Figure 19).



[Schach]

Figure 19 Comparison of UML to Concept Maps

In Figure 19 the left hand figure is a UML representation of the elevator problem presented in the tutorial [Schach]. The right hand side is the concept map generated from the same problem. It is the intention of this work to show that the figure on the right is easier for stakeholders to use as a common communications tool due to its combination of natural language and graphical features.

BIBLIOGRAPHY

- Anderson, L., et al. *Using Concept Mapping to Develop a Logic Model for the Prevention Research Centers Program*. Preventing Chronic Disease Public Health Research, Practice, and Policy, Volume 3, Number 1. January, 2006.
- Ausubel, D.P., *The use of advance organizers in the learning and retention of meaningful verbal material*. Journal of Educational Psychology, 51. 1960.
- Bass, L., Klein, M., Bachmann, F., *Quality Attribute Design Primitives*. Carnegie Mellon University, 2001.
- Bass, L., Klein, M., Bachmann, F., *Quality Attribute Design Primitives*. Carnegie Mellon University, 2001.
- Booch, G., Rumbaugh, J., Jacobson, I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.
- Brooks, F. P., . *The Mythical Man Month*. Addison Wesley, 1995.
- Bruillard, E., Baron, G.-L.; *Computer-Based Concept Mapping: a Review of a Cognitive Tool for Students*. In: Proceedings of Conference on Educational Uses of Information and Communication Technologies (ICEUT), 2000.
- Buhr, R.J.A.. *Understanding Large-Scale Behaviour in Complex Systems*. In: Proceedings, Second IEEE International Conference on Engineering of Complex Computer Systems, 1996.
- Cederling, U.; Ekinge, R.; Lennartsson, B.; Taxen, L.; Wedlund, T.. *A project management model based on shared understanding System Sciences*. Proceedings of the 33rd Annual Hawaii International Conference on, Vol., Iss., 4-7, 2000.
- Christel, M. G., Kang, K. C., . *Issues in Requirements Elicitation*. Carnegie Mellon University, Software Engineering Institute, 1992.
- Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., Stafford, J.; *Documenting Software Architectures; Views and Beyond*. Addison Wesley, 2003.
- d'Auriol, Brian J., *A Concept Visualization Study of a Parallel Computing Program*. In: Proceedings of the Third International Conference on Requirements Engineering, 1998.
- Darmofal, D.L.; Soderholm, D.H.; Brodeur, D.R.; *Using concept maps and concept questions to enhance conceptual understanding*. Frontiers in Education, 2002.
- Denning, Stephen. *The Leader's Guide to Storytelling*. John Wiley and Sons, 2005.
- Deutsch, Michael S., *Focusing Real-Time Systems Analysis on Operations*. IEEE Software, September 1988.
- Eden, Amon H., *A Theory of Object-Oriented Design*. Information Systems Frontiers 4:4 379-391, Kluwer Academic Publishers. 2002.
- Ferguson. *Object-oriented Concept Mapping using UML class diagrams*. Journal of Computing Sciences Volume 18 , Issue 4, 2003.
- Freeman, Lee A., *The Effects of Concept Maps on Requirements Elicitation and*

- System Models During Information Systems Development*. In: Proceedings of the First International Conference on Concept Mapping, 2004.
- Gay, G., Hembrooke, H.. *Activity-centered design : an ecological approach to designing smart tools and usable systems*. MIT Press, 2004.
- Genero, M., Piattini, M., Calero, C., *Metrics for Software Conceptual Models*. World Scientific Publishing Company, 2005.
- Grinter, R. E.. *Systems architecture: product designing and social engineering*. In Proceedings of the international Joint Conference on Work Activities Coordination and Collaboration, February 22 - 25, 1999.
- IEEE.EIA. *Industry Implementation of International Standard ISO/IEC 12207 : 1995 (IOS/IEC 12207) Standard for Information Technology – Life cycle data*. IEEE and EIA, April, 1998.
- IEEE.EIA. *Industry Implementation of International Standard ISO/IEC 12207 : 1995 (IOS/IEC 12207) Standard for Information Technology – Software life cycle possesses*. IEEE and EIA, March, 1998.
- Jain, H., Vitharana, P., Zahedi, F., *An Assessment Model for Requirements Identification in Component-Based Software Development*. In: Advances in Information Systems, Fall 2003.
- Knight, Claire. *Visualisation for Program Comprehension: Information and Issues*. Department of Computer Science, University of Durham, 1998.
- Kop, C., Mayr, Heinrich. *Conceptual Predesign Bridging the Gap between Requirements and Conceptual Design*. In: Proceedings Third International Conference on Requirements Engineering, 1998.
- Kramer, S.; *Application of concept mapping to systems engineering*. In: Proceedings of , Systems, Man and Cybernetics, 1990.
- Kremer, R., Gaines, B. R.; *Groupware Concept Mapping Techniques*. In: Proceedings of the 12th annual international conference on Systems documentation, 1994.
- Maurer, S. B.; Ralston, A.. *Discrete Algorithmic Mathematics, Third Edition*. A K Peters, 2004.
- Navarro, L.I.; Such, M.M.; Martin, D.M.; Sancho, C.P.; Peco, P.P.; *Concept maps and learning objects*. , Fifth IEEE International Conference on Advanced Learning Technologies, 2005.
- Novak, J., *Applying learning psychology and philosophy to biology teaching*. The American Biology Teacher, 43(1). 1981.
- Novak, J., *Learning, Creating, and Using Knowledge: Concept Maps™ as Facilitative Tools in Schools and Corporations*. LEA Incorporated. 1998.
- Parnas, D. L., Weiss, D. M.. *Active Design Reviews: Principles and Practice*. Journal System Software, Vol. 7, Num. 4. 1987.
- Pinch, T., Bijker, W.. *The social construction of facts and artifacts: or how the sociology of science and sociology*

- of technology might benefit each other.*
In *Social Studies of Science* (Vol. 14, pp. 399-441). London: SAGE, 1984.
- Preparata, F. P. and Yeh, R. T..
Introduction to Discrete Structures for Computer Science and Engineering.
Addison-Wesley Longman Publishing Co., Inc., 1973.
- Schach, Stephen. *Classical and Object-Oriented Software Engineering.*
McGraw-Hill, 1999.
- Schuler & A. Namioka (Eds.).
Participatory design: principles and practices. L. Erlbaum Associates, 1993.
- Siau, K., Tan, X.; *Technical Communication in Information Systems Development: The use of cognitive mapping.* IEEE Transactions on Professional Communication, Vol. 48, No. 3, 2005.
- Smolander, K., Pivrinta, T.. *Describing and Communicating Software Architecture in Practice: Observations on Stakeholders and Rationale.*
Proceedings of CAiSE'02 - The Fourteenth International Conference on Advanced Information Systems Engineering, Toronto, Canada, May 27 - 31, 2002.
- Turns, J.; Atman, C.J.; Adams, R..
Concept maps for engineering education: a cognitively motivated tool supporting varied assessment functions. IEEE Transactions on Education Volume: 43 Issue: 2, 2000.
- Walker, G., *Challenges in Information Visualisation.* In: British Telecommunications Engineering Journal, Vol. 14, pp 17-25, April 1995.

