OPEN UNIVERSITY OF THE NETHERLANDS

MASTER THESIS

8 MAY 2018

Improve software design understanding

Ivan Timmers

supervised by
Dr. ir. H. PASSIER
Dr. ir. S. STUURMAN

Contents

List of Figures

SUMMARY

Problem The last decade, software projects increased significantly in size. It becomes increasingly difficult to comprehend the software. Understanding the structure of the software can be hard and time consuming. Design documentation is often available but most of the time out of date and obsolete. Many software engineers rely therefore on software visualization tools that help in software understanding. In combination with the initial documentation the intent of the software can become clear.

When software becomes larger, the traditional visual tools for displaying software design contain issues.

- The UML visualization technique does not scale well with large diagrams. The diagrams become cluttered with information like entities and relationships. Also 2D inheritance trees create space filling diagrams that are hard to read.
- Layout techniques with 2D entities are hard. With large diagrams the numbers of lines and crossings can grow very quickly.
- Navigation tool like pan-zoom and overlapping windows cause for getting lost in detail. The overlapping windows causing discontinuity problems.
- Different diagrams are created for the same project. Consistency is often lost.

Vision Our vision is to improve the understandability of software design. We acknowledge that this is a long-term vision, which should not be considered lightly. Our short-term vision however is to deliver a software tool that shows how software design understandability can be improved. We believe that 3D techniques and the coupling of source-code metrics to UML class diagrams can contribute to our vision.

Method We created a prototype software tool that implements possible answers to the problems mentioned above. The problems are divided into three domains: visualization, navigation and filtering. For each of these domains we implemented solutions in our software tool. To make the added value visible we compared existing documentation with the documentation generated by our tool. We had the opportunity to use student exercises that was already examined by tutors. We compared the results from the first rating with the rating based on our software tool.

Techniques To improve the visualization we implemented a way to convert existing 2D UML diagrams to 3D. We use the XMI exchange format to load the UML elements in a 3D engine created with Unity. The UML elements are visualized in 3D. Besides the UML

model we made it possible to add source-code information in a 3D metric view. The metric information is combined with the UML diagram.

Navigation controls are implemented to navigate through the model. We implemented controls as found in Computer Aided Design software. Filtering options are added, for example, filtering by name or type. The coupling with the source-code metrics is made. Interesting elements in the metric view are highlighted in the UML diagram and vice verse. Also a filter based on changes in the UML diagram is created.

Results Based on the validation results we can conclude that the conversion of 2D UML to 3D UML diagrams does not directly give added value. But when combining the UML diagrams with source-code metric information in 3D the added value can clearly be seen. When validating the results from the students, focus points where identified which were not seen with the traditional 2D UML class diagrams.

SAMENVATTING

Probleem De laatste decennia zijn software projecten voortdurend gegroeid in grootte. Het wordt steeds moeilijker om de software te doorgronden. Het begrijpen van de structuur van de software is vaak moeilijk en tijdsintensief. Ontwerp-documentatie is vaak beschikbaar maar deze is meestal verouderd. Veel software engineers vertrouwen hierom op software visualisatie tools die helpen om de software te begrijpen. In combinatie met de documentatie kan de structuur en werking van de software duidelijk worden. Wanneer software projecten groter worden, bevatten de traditionele visuele tool die software-ontwerpen visueel maken problemen:

- De UML visualisatie techniek schaalt niet goed met grote diagrammen. De diagrammen zien er rommelig uit met veel informatie zoals entiteiten en relaties. Ook 2D overerving-structuren nemen erg veel plaats in die moeilijk te lezen zijn.
- Efficiënte lay-out technieken met 2D elementen zijn lastig. Voor grote diagrammen kunnen de aantal lijnen en lijn kruisingen erg hard groeien.
- Navigatie technieken zoals pan-zoom en overlappende vensters zorgen ervoor dat de gebruiker verdwaald in de details van het diagram. Overlappende vensters veroorzaken daarnaast discontinuïteit problemen.
- Verschillende diagrammen worden gemaakt voor hetzelfde project. Consistentie gaat hierdoor vaak verloren.

Visie Onze visie is om de begrijpelijkheid van software ontwerp te verbeteren. We erkennen dat dit een lange-termijn visie is, die niet lichtvaardig moet worden beschouwd. Onze korte termijn visie is daarom om een software tool op te leveren die laat zien hoe de software begrijpelijkheid van worden verbeterd. Wij geloven dat 3D technieken en de koppeling van source-code metrieken aan UML klasse diagrammen kunnen bijdragen aan onze visie

Methode We hebben een prototype software tool ontwikkeld die mogelijke antwoorden implementeert op de problemen zoals hierboven beschreven. De problemen zijn opgedeeld in drie domeinen: visualisatie, navigatie en filteren. Voor elke van deze domeinen hebben we oplossingen geïmplementeerd in onze software tool. Om de toegevoegde waarde zichtbaar te maken hebben we bestaande documentatie vergeleken met de documentatie gegenereerd door onze tool. We hadden de kans om opdrachten van studenten te gebruiken die al waren beoordeeld door docenten. We hebben de resultaten vergeleken van de eerste beoordeling met de beoordeling gebaseerd op onze software tool.

Technieken Om de visualisatie te verbeteren hebben we een manier geïmplementeerd om bestaande 2D UML diagrammen te converteren naar 3D. We hebben gebruik gemaakt van het XMI uitwisseling-formaat om de UML elementen in een 3D engine te laden gemaakt in de Unity omgeving. Naast het UML model hebben we het mogelijk gemaakt om source-code informatie toe te voegen in een 3D metrieken overzicht.

Navigatie mogelijkheden zijn geïmplementeerd om door het model te navigeren. Deze mogelijkheden kunnen ook gevonden worden in Computer Aided Design software. Filter opties zijn toegevoegd, zoals filteren op naam of type. Ook koppeling met de source-code metrieken is geïmplementeerd. Interessante elementen in het metrieken overzicht worden benadrukt in het UML diagram en andersom. Ook is er een filter toegevoegd gebaseerd op veranderingen tussen verschillende versies van het UML diagram.

Results Gebaseerd op de validatie resultaten kunnen we concluderen dat de conversie van 2D UML naar 3D UML diagrammen niet direct een toegevoegde waarde geeft. Wanneer de UML diagrammen gecombineerd worden met de source-code metrieken in 3D, wordt de toegevoegde waarde wel duidelijk zichtbaar. Met het valideren van de resultaten van de studenten, werden aandachtspunten duidelijk geïdentificeerd die niet duidelijk konden worden gezien met de traditionele 2D UML klasse diagrammen.

1

INTRODUCTION

Throughout a software product's life cycle, many people should be able to understand the software code. Learning the structure of software code is hard and can be time consuming. When design documentation is available this can help, but however, after several maintenance cycles these documents tend to be out of date and are often obsolete. Many software engineers therefore rely on software visualization tools that help in software understanding. In combination with the initial documentation, the intent of the software can become clear.

Documentation During the development phase of a software program several techniques are used to document the software, as for example, behavioral diagrams such as call-graphs, or models like UML diagrams. Graphical languages like this provide users with a higher level of abstraction of the software [1]. Together with the software metrics the documentation holds a lot of information about the software code. Users have access to different tools that contain different data. Code metrics are used to say something about the quality of the software, while for example UML domain diagrams describe architectural information.

UML class diagrams For software design, UML diagrams are widely used to document software. One of the diagrams specified by the UML standard is the class diagram. This diagram is used during and after the requirements phase to model the domain logic, and also as design model. The use of 2D diagrams is suggested in the literature [6] to provide a higher level of abstraction of the software under investigation. The purpose of these tools is to enable the user to comprehend structure and relationships through visual mappings.

Problems These mapping tools however, have limitations when dealing with large diagrams. Although UML diagrams have advantages in comparison to a situation without visualization, UML visualizations have shortcomings. Especially when dealing with large software projects the current visualization techniques tend to be difficult to comprehend due to various reasons [13, 14, 5]. The following reasons can be identified for dealing with a difficult to comprehend documentation technique. These reasons are further described in chapter 3.

- The UML visualization technique does not scale well with large diagrams. The diagrams become cluttered with information like entities and relationships. Also 2D inheritance trees create space filling diagrams that are hard to read [14, 13].
- Layout techniques with 2D entities are hard. With large diagrams the numbers of lines and crossings can grow very quickly [14, 13].
- Navigation tool like pan-zoom and overlapping windows cause for getting lost in detail. The overlapping windows causing discontinuity problems [14, 13].
- Different diagrams are created for the same project. Consistency is often lost [14, 13].

Several papers offer solutions for visualization systems that help understanding a complex system [14, 13, 17]. These solutions mostly cover the metric visualizations and reverse engineering techniques. Metric information gives quality information of aspects of the source-code. These information must be manually mapped to the domain model. This introduces a new problem with the current visualization techniques:

- Source-code information must be manually mapped to the domain model.

3D visualizations can help overcome the first three problem areas as described above. It creates a physical object, a visual, that represents the software system and gives software engineers insight on the complex software structure. Previous work has shown that, for at least UML state diagrams, 3D visualization prove useful [11]. We claim that 3D visualization can be useful for other types of visualization to improve the software understandability.

The consistency problem cannot be solved with 3D visualization and is not investigated in this research project.

Vision Our long-term vision about software visualization is to come with a solution to improve software understanding with 3D visualization of UML class diagrams and metric information. Class diagrams give information about the design, but no information about how the software functions in code. The combination will be made between UML class diagrams and software metrics to combine the results from both worlds into one view. One single view for design *and* for code quality.

In short-time we will come with a prototype software tool that proves that the long-term vision can be realized technically speaking. To do this we divide the 2D UML problems in the following area‘s to solve [8]:

- Visualization
- Navigation
- Filtering

These problem area‘s are discussed in chapter 3

1.1. CONTEXT & SCOPE

This research focuses on visualization techniques of UML class diagrams and metrics visualization. UML class diagrams are used by software engineers in a UML domain diagrams and in a UML design diagram. Several software development processes make use of the UML class diagrams during one or more phases [7].

The visualization techniques are used by software engineers to get insights in the software code. These users possible profit from the solution proposed in this research.

Education The validation of the research will take place within the higher education field. In education, and especially software engineering, a lot of software designs has to be created and rated by instructors. Getting insight in the design is an important component when examining a design. Our software tooling will help the instructor and the student to get better insight in their designs and add information about the quality of the design with metrics.

Scope Dynamic software behavior described in dynamic UML diagrams like sequence diagrams are not part of this research. A prototype tool will be developed that uses 3D techniques to overcome the problem areas of 2D UML class diagrams and combine them with software metrics. Two metrics are used to demonstrate the prototype: number of attributes and number of methods in a class.

1.1.1. SOFTWARE DESIGN

Most of the time software design work during the analysis and design phase is done with UML diagrams. After the requirements for the current iteration a domain model is created with a UML class diagram notation. When dealing with UML class diagrams, with the current tools there is no connection between the UML class diagram and the code metrics. Code metrics tooling is often used in collaboration with the UML class diagrams for references. Our research focuses on the software design phase.

When a change is made in a UML class diagram it can be hard to see what the impact is on the quality of the source-code. An extra feature of the 3D prototype tool that we developed during this research is that it is possible to compare 2 UML diagrams for changes. The changes are clearly shown on screen. The current metrics are shown with the UML diagram. This makes it possible to see the impact of changes from the UML class diagram to the code quality with metrics.

1.2. THESIS OVERVIEW

The following topics are covered in this master thesis. Chapter 2 describes the research method and the method that is used to validate the research questions. Chapter 3 provides an overview of the literature about problems with 2D visualizations, 3D UML representations and software metrics in a background section. Chapter 4 shows the results of the software tool that is created. In chapter 5 the validation results are discussed. The last chapter contains the conclusions of this research and proposals for future work.

2

RESEARCH METHOD

This chapter describes the research method used and the validation procedure.

2.1. INTRODUCTION

To break down our long-term vision, to improve software understanding with 3D visualization, into smaller steps we did research a piece of the complete puzzle. We call this the short-term solution.

This first step in this short-term solution will deliver a prototype tool that shows that it is possible to use 3D techniques for displaying UML class diagrams and metrics. With the results we can answer a piece of the long-term question. Future work can continue with these results to come with a complete solution.

To answer the research questions a prototype software tool is created. This is necessary to verify the claims done in this paper. To validate the added value of this tool and thus the added value of this research a validation procedure has been created. The course "Design Patterns" within the software engineering master on the Open University will be used as validation context.

2.1.1. VALIDATION CONTEXT

During the master program "Software engineering" on the Open University one of the courses is about software design patterns. A substantial part of this course is to refactor an existing Java application called "Jabberpoint" to an application that makes use of design patterns. The students work in a group of 2 on the same assignment.

We had the opportunity to use existing results of some student teams as input for our software prototype tool. As of this moment students delivering results in the form of UML class diagrams drawn in different tools with a lot of documentation and the complete Java source code. These UML diagrams are a lot to handle for the instructors of the course. Sometimes it can be difficult to see if an UML class diagram is correct or not without deeply examine the files and combine the UML class diagrams with the source code. At the moment the software metrics are not included in the examination of the results of the students.

The results are already rated by the instructors based on the 2D UML class diagrams. We take the examined work and convert the diagrams and source-code with our prototype tool and deliver a 3D representation of the same work. This way we could examine the difference in perceptions of the quality of the results.

2.2. RESEARCH METHOD

The goal of this research is to deliver a prototype tool that can be used to get information about UML class diagram designs. This tool can be used in addition to 2D UML tools to retrieve extra information about the software design. That makes this research a constructive research: a prototype is made and tested.

2.3. RESEARCH QUESTIONS

The research goal is to prove that 3D techniques with UML have an added value to existing 2D UML representations. The goal is to make the software design phase more easy to understand.
The **research question** is:

How can we add value to software understanding with 3D techniques?

Sub-questions are:

- How can 3D, compared to 2D, help by understanding software design?
- How can navigation controls be implemented in 3D UML class diagrams?
- How can filter possibilities be implemented in 3D UML class diagrams?
- How can source-code metrics be combined with 3D UML class diagrams?

Method First, to answer the questions, we conduct a literature study of different visualization techniques for UML and metrics. The result of this study is a set of rules about transferring from 2D to 3D and the current difficulties with 2D UML diagrams.

Also more insight is gained about the current problems with 2D UML diagrams and the research sub-questions are formed on basis of this study. The results of the pre-study are used as a base to create a prototype to investigate how the research questions can be answered.

The research question that must be answered cannot be measured in the form of a metric. For example, the added value of the software tool can be expressed by the degree of information overload. But this cannot be measured and is highly dependent of the person. Therefore the validation is done by examining the added value of the software prototype tool within the context as described above. The validation procedure will be done by different persons to check for consistency.

We chose this validation method because of the current state of the software that we developed. The software is developed as prototype. This makes the tool unsuitable to test

with a group of test persons. First the useability must be improved. The validation method as described is compact and is expected to grant usable results.

2.4. VALIDATION

2.4.1. SCOPE

The validation method focuses on our short-term goal: creating a prototype tool that proves the described problems in section 1 can be solved, technically speaking. We limit the scope therefore to create a prototype tool with the following functions:

- Display a 3D UML design class diagram.
- Intuitive navigation controls.
- Multiple filtering options.
- Visualize metrics.
- Combine metrics with UML class diagrams.

Useability requirements are not included in this research.

2.4.2. VALIDATION METHOD

The expectation about the prototype software tool is that 2D UML diagrams visualized in 3D increase the understanding of the diagram. With the help of software metrics, it should be easier for the instructor to make a decision about the quality of the work of the student. The method that we used is to take different submitted assignments of students and convert them to a format our software tool can handle. This includes:

- Remake the 2D UML diagrams in a UML tool and export the diagram to XMI format.
- Extract the software metrics from the source code.
- Import the XMI file in the 3D software tool.
- Position the 3D UML entities in a 3D world.
- Import the software metrics as 3D city metaphor coupled to the 3D UML diagram.

This is done for 3 different projects. The result of this transition is a 3D workspace with the 3D UML diagram and the metric information. The resulting workspace is given to the instructors of the course and they write down their findings about this tool. Alongside with our own findings, the research questions can then be answered.

When validating, the differences must be addressed between examining the results of the student with the 2D diagrams and the 3D conversions. The following aspects are important:

- Is the positioning of the 3D entities found convenient?

- Is the metric information of added value in addition to the UML class diagram?
- Does the navigation techniques implemented helps with understanding the model?
- Do the navigation problems reduce with 3D diagrams?
- Do the filtering techniques help to emphasize key areas of the model without losing track?
- Does the coupling between metrics and the UML class diagrams help by identifying problem area‘s?
- Are the same amount of faults found when examining the 3D results as with the 2D results?

The results of the validation procedure can be used to answer the research questions.

2.5. RESEARCH CONTRIBUTIONS

The current research area focuses on displaying technical qualities of the software code-base with help of code-metrics to reverse-engineer the software properties. Not often this is related to the design phase in software to cross-link the metrics information with the design information.

The first contribution of this research is to deliver a method to transfer existing 2D UML diagrams into a 3D representation. The UML notation is preserved in 3D space, as the UML entities are converted to 3D counterparts. In the 3D space these entities can be moved to a position that is most valuable for the viewer. Because of the 3D space more entities can be displayed then in the 2D space with less line-crossings. Also some of the navigation problems can be solved. This work has been done before [11, 5], but we make use of a modern standard 3D engine that has a great performance boost over the engines used before. The use of a standard engine will also make it easier to extend in the future.

Second, to make the bridge to the code-base, a way is implemented to read an existing source-code file and extract the metrics. These metrics can be displayed as a variant of the codecity metaphor. The codecity is also displayed in 3D and a relation will be set between the 3D UML and the 3D codecity. This makes it possible to highlight different parts of the software in the design view and highlight the corresponding parts in the metric view. Additionally the differences between the design and code are made visible, to be able to verify if the code complies with the design.

Although we found one research about the coupling of UML and metrics [15], this is a different approach as our research. We combine the added value of metric metaphors and 3D UML designs, instead of viewing 3D metric representations on 2D UML diagrams. The expectation is that the time to understand a software design will greatly improve due to the coupling of both domains.

Overall, the short-term contribution is a prototype tool that transforms existing 2D UML class diagrams and source-code to a 3D UML class diagram with a codecity metric view. This gives insight in how the long-term solution, improve software design understanding, might be solved. The coupling of metric information to 3D UML diagrams has not been done before.

3

BACKGROUND

Here, we give an overview of the background of software visualizations.

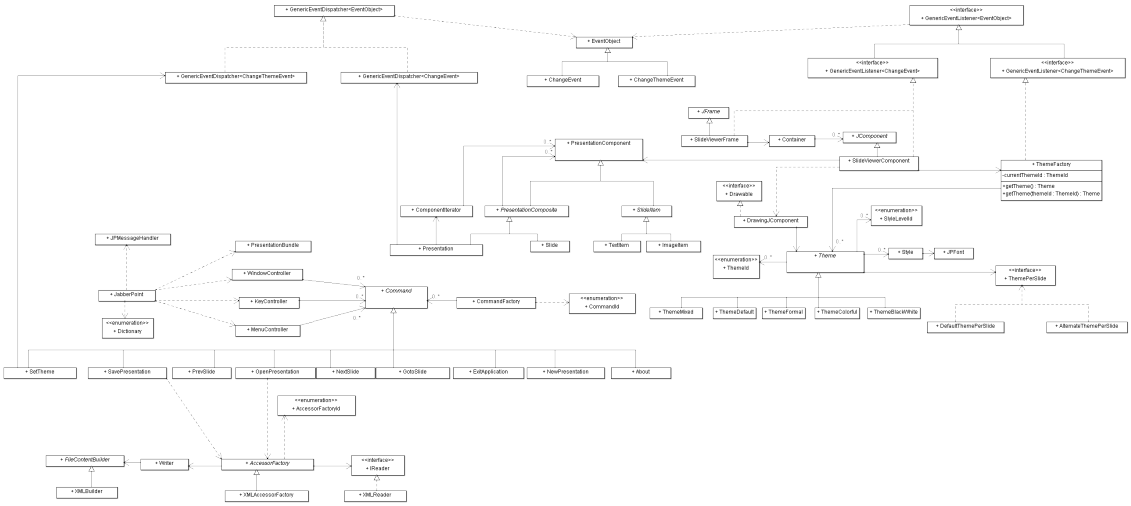
3.1. 2D VISUALIZATION

In software design, UML class diagrams are often used to visualize software design [10, 14, 5, 11]. Also other forms of 2D visualization have been suggested [1]. These diagrams provide users with a higher abstract view of the software. The main goal is to simplify program comprehension with diagrams that enable the user to comprehend complex internal structure and relationships. This is done by visual mapping. These 2D mapping tools however, have limitations when dealing with large diagrams. Although 2D diagrams have advantages in comparison to a situation without visualization, 2D visualizations have shortcomings [14, 13, 5].

- UML class diagrams do not scale well with more complex designs. The diagrams become cluttered with information. Also, 2D inheritance trees create large diagrams that are hard to read.
- The task to create a clear layout techniques for 2D class diagrams is hard. With larger diagrams, the numbers of lines and crossings grows very fast.
- The use of navigation tools like pan-zoom and overlapping windows, that are needed with large diagrams because it would be impossible to read a class diagram on a screen otherwise, causes the user to get lost in details. Overlapping windows cause discontinuity problems.
- Several diagrams may be created for the same project. Consistency is often lost, when not using a tool that automatically checks for consistency.

Software applications become increasingly complex and large. Therefore, having a visualization technique that scales with the size of the diagrams becomes more important. Figure 3.1 shows an example of a large UML class diagram of a real-world example. As can be imagined, the larger the diagram, the more difficult it is to understand the diagram.

Figure 3.1: Large 2D UML class diagram



It is also more difficult to find a clear layout for large diagrams. Large diagrams contain more entities and more relationships, which increases the chance of line crossings, which make the diagram less readable. A solution is to split the diagram into multiple pages and switch between them, but this can confuse the reader [11]. Also, it is harder to maintain consistency without the support of a tool that automatically checks for inconsistency

3.1.1. PROBLEM AREA 2D VISUALIZATION

The problems we sketched can be divided into three categories [8]:

Visualization Large, complex diagrams are often subdivided into more diagrams, that can be viewed in several windows (open in tabs). It is difficult to understand large complex diagrams this way. For example, large diagrams can have several hundreds of references in the hierarchy. One sub-component can be referenced by several other components. This is not visible in 2D without continuously switching between windows.

Navigation Often in 2D UML tools one diagram is visible per window, which puts limits on the navigation through the hierarchy. This ensures that a user always has to switch back and forth between different views. This limits the comprehension of complex models [8]. This is indicated as “one easily can loose overview of the model, not remembering the structure and design of the model”. These results support the facts that 2D navigation techniques lack in supporting the user with memorizing the model.

Filtering Information overload can occur by users of current 2D UML tools due to the lack of filtering possibilities. Complex diagrams cannot be dynamically filtered to only show the parts of the diagram that the user is currently interested in.

The next section describe how 3D techniques can be used to overcome these problems.

3.2. 3D VISUALIZATION

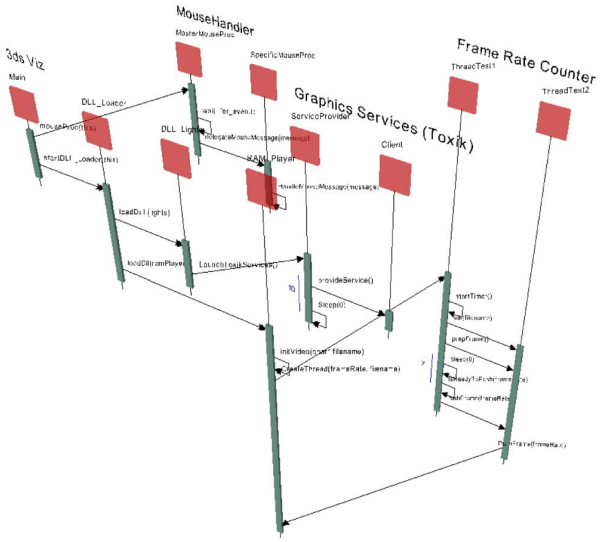
3D visualizations can help overcome the problem areas as described above. It creates a physical object, a visual, that represents the software system and gives software engineers insight on the complex software structure.

3.2.1. WHY 3D?

The first question rises is why convert 2D to 3D? The use of 2D diagrams is suggested in the literature [6] to provide a higher level of abstraction of the software under investigation. The purpose of these tools is to enable the user to comprehend structure and relationships through visuals mappings. Despite 2D diagrams add a lot more comprehension this is less the case for large systems. It becomes increasingly difficult to comprehend large diagrams due to the reasons mentioned in section 1 on page 1

Visualization techniques as 2D UML class diagrams use abstract models that represent concrete software source-code. A model is thus used to represent code. This is called a metaphor. A metaphor is used to understand one specific thing using the terms of another thing [4]. For example an UML class diagram uses squares for classes that represents concrete entities. Additional the coupling between code entities is modeled with line metaphors.

Figure 3.2: 3D UML



In the last two decades, programs have become larger and more complex. Programmers tend to have increasing difficulties to visualize these complex and large code structures. A few pre-defined views may not be sufficient when dealing with very large structures. Also not every visualization technique can display all aspects of the software in the best way [12]. This is why we conducted a research to the use of 3D instead of 2D. In the literature often 3D visual representations are suggested as a solution to help overcome these problems[11, 5, 14, 13].

There are surveys that clarify the use of 3D techniques for displaying software diagrams at least with state diagrams to bring advantages [11]. But not only in state diagrams, the

advantages of 3D are initially obvious. The amount of information perceived is increased with a certain factor [5]. For example in hierarchical or layered structure, components of the same level implies linear display in 2D consuming a lot of space. In 3D such data can be laid out on a plane in the third dimension.

But this is only the case if effective use is made of the added dimension. Most of the current approaches are just transforming 2D visualization techniques into a 3D space. Although the use of 3D offers a greater working area, also they introduce readability issues. Problems that might be introduced by 3D visualization techniques include objects being obscured, disorientation and spatial complexity. To some extend these issues can be solved by interaction techniques that change the viewpoint of the 3D graph within the diagram. Otherwise the 3D representation is just limited to a 2D picture of a 3D structure. An example of a 3D technique used with no added value to 2D can be seen in figure 3.2

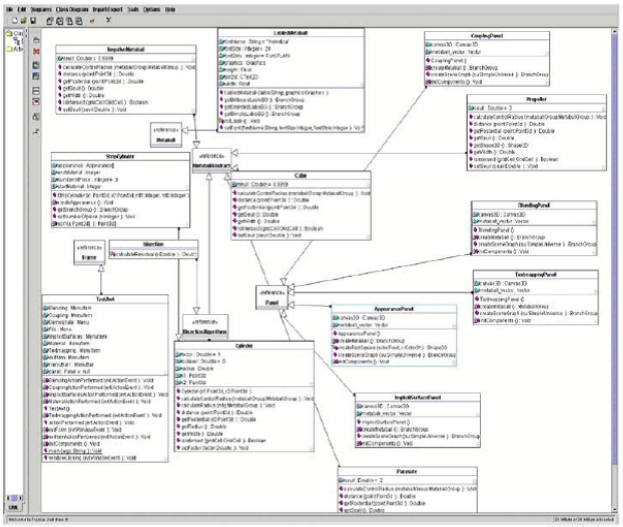
3.3. SOFTWARE METRICS

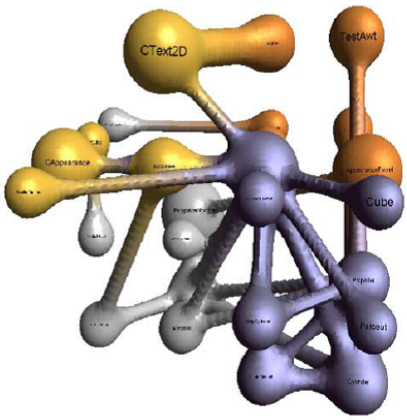
What is a software metric? In literature, software measure, software measurement and software metrics are often used interchangeably. IEEE does provide the following definition: *"The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software."*.

The use of metrics has been common use in the software engineering world. It has been shown that software metrics can improve understandability of a design and code. Software metrics also contribute to its future maintainability. [2]

In software visualization many examples can be found where metaphors are used [17, 13]. In figure 3.3 an example can be seen where a 2D UML diagram is transferred to a 3D picture with the use of a metaphor. The purpose here is to display specific parts of the software metrics like coupling and cohesion.

Figure 3.3: Metaball metaphor

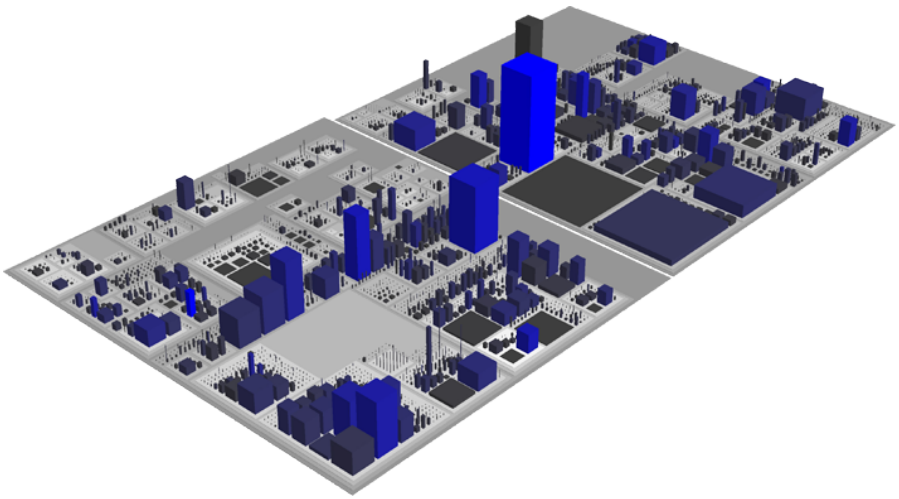




The metaphor used in this example is called “Metaball” metaphor. The researchers distinguished two types of metrics, Cohesion (internal aspects) and Coupling (structural aspects). This research shows that added value is gained by combining different domains in one overview.

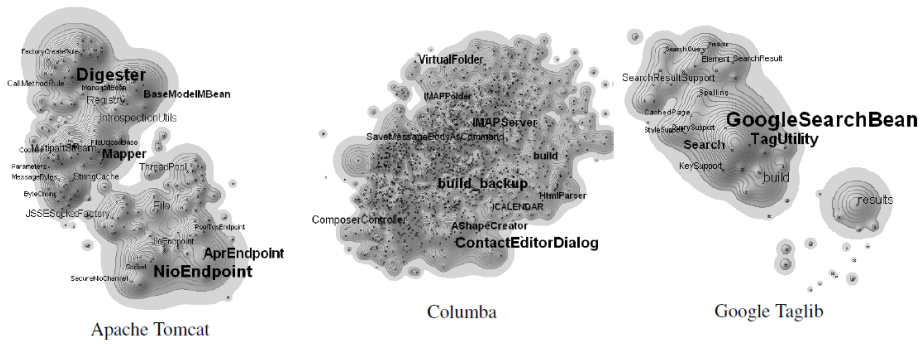
Another metaphor mentioned in the literature is the codecity metaphor. This metaphor describes the software metrics with the help of a city metaphor. Two kinds of metrics are taken into account: number of attributes (NOA) and number of methods (NOM). The NOA is used to define a building surface. The NOM defines the building height. See for an example figure 3.4. The prototype software implements a variation of this city metaphor.

Figure 3.4: Codecity metaphor



Landscapes are also mentioned as a possible metric metaphor [9]. An example of the landscape metaphor can be seen in figure 3.5. The authors claim that since software has no physical shape, there is no ”natural“ mapping of software to a two-dimensional space. A consistent layout can be generated from the software metric that allows users to develop a variety of thematic software maps that express very different aspects of software while making it easy to compare them. Landscapes allow users to achieve a consistent view off the software. The shape of the landscape is formed by the lexical similarity of the source code. The height of the hills are calculated with the KLOC (thousands of lines of code) metric.

Figure 3.5: Landscape metaphor

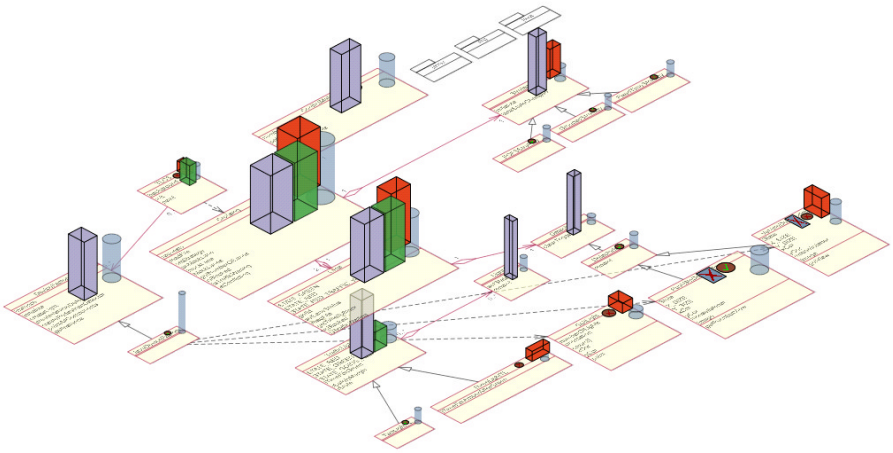


3.4. COMBINE UML AND METRICS

UML class models are usually created and used as an interactive modeling tool. Metrics on the other hand are effective for analyzing large system architectures. Metrics can answer complex questions that cannot be answered with UML class diagrams. To reduce the cognitive disruption caused by switching between multiple views, it is desirable to combine the domains of UML class diagrams and metrics in one view.

At least one research has been done on the combination of UML class diagrams with metric information [15]. A tool called "MetricView" is essentially an UML visual tool that adds highly customizable metric visualizations. A given UML class diagram and a set of metrics can be combined in one view. An example can be seen in figure 3.6

Figure 3.6: MetricView example



The research shows promising results for combining both the strengths of UML class diagrams and metric data. Our prototype software tool presented in the next section continues on their conclusion that the combination of UML class diagrams and metric information is an added value over the existing visualization techniques. Because the visualization technique of the UML class diagram in "MetricView" does not include a 3D representation, our research did implement a 3D variation of the UML class diagram and a 3D metric view.

3.5. FUNCTIONAL REQUIREMENTS

Based on this background study possible requirements are formulated. The next chapter will continue on the requirements that are implemented in this research with a prototype software tool.

The functional requirements are the requirements that came forth of our research-goal. The goal is to improve UML class diagram visualization. We think that 3D diagrams are a part of the solution to the problems. We only focus on the functional requirements in this research study. The prototype software tooling must meet the following functional requirements:

1. XMI format must be supported for importing UML diagrams.

2. A UML class diagram must be displayable in 3D.
3. Convenient 3D navigation controls must be implemented.
4. It must be possible to display code-metric information in 3D.
5. Different filter possibilities must be implemented to filter the UML entities.
6. It must be able to combine 3D Metric information with the UML class diagram.

These functional requirement are explained in detail in the next chapter.

4

SOFTWARE PROTOTYPE

To answer the research questions, a prototype tool is developed. This tool is able to import an existing 2D UML diagram in the XMI format and converting the diagram into a 3D representation. Navigation controls are implemented to inspect the model. Also filtering options are implemented In 3.1.1 on page 9 problems with 2D UML class diagrams are mentioned. The software tool is developed with these problems in mind and is presented in the next section.

4.1. FUNCTIONAL REQUIREMENTS

Not all functional requirement from 3.5 on page 13 are implemented in the prototype tool. This section gives an overview of the implemented functional requirements and how this is realized.

The functional requirements that we have implemented for our short-term goal is based on the current problems for 2D UML class diagrams. The problems are here mentioned again:

- UML class diagrams do not scale well with more complex designs. The diagrams become cluttered with information. Also, 2D inheritance trees create large diagrams that are hard to read.
- The task to create a clear layout techniques for 2D class diagrams is hard. With larger diagrams, the numbers of lines and crossings grows very fast.
- The use of navigation tools like pan-zoom and overlapping windows, that are needed with large diagrams because it would be impossible to read a class diagram on a screen otherwise, causes the user to get lost in details. Overlapping windows cause discontinuity problems.
- Several diagrams may be created for the same project. Consistency is often lost, when not using a tool that automatically checks for consistency.

The first and second problem is tackled by implementing 3D UML diagrams to increase the scalability and drawing space. Also filter possibilities are implemented to help the user to filter out irrelevant data. The filter options include an option to view the source-code metrics with the UML data to filter the software design on the basis of source-code metrics.

The third problem is solved by implementing navigation tools in 3D. This allows the user to explore the model in an extra third dimension that removes the overlapping windows.

The inconsistency problem is not solved in the research.

4.2. CONSTRAINS

For this prototype there has been chosen for certain tools and programs to work with. This section gives an motivation about the choices.

- ArgoUML
  ArgoUML has been chosen as UML modeling tool. This tool is able to generate XMI files. XMI stands for XML Metadata Interchange. The format of XMI is maintained by the Object Management Group. XMI describes a XML-format, which can be used to exchange data of UML-diagrams between tools. Although XMI is a standard, a XMI-file generated by ArgoUML cannot be read, for example, by the UML modeling tool Visual Paradigm. The XMI-file is processed by a XML parser which is created using the .NET Framework.

- Unity and C#
  Unity is chosen as 3D engine to create the tool. Unity is a cross-platform game engine, which is primarily used to develop video games and simulations for computers, consoles and mobile devices. Unity uses C# as scripting language. C# runs on the Microsoft .NET framework. We had previous experience with Unity and C#, therefore there has been chosen for these techniques to develop the tool with.

4.3. TECHNICAL IMPLEMENTATION

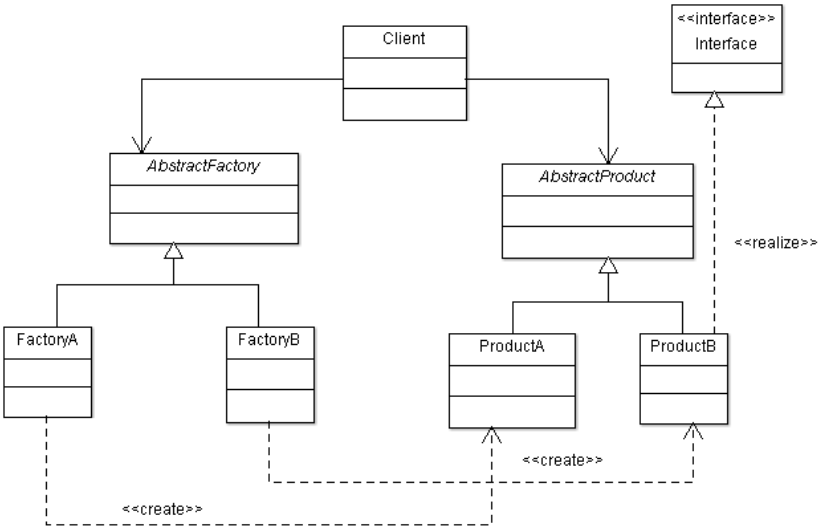
This section describes how the functional requirements from section 3.5 are implemented in the prototype tool. For demonstrating purposes we use a simple non-functional UML diagram that is shown in figure 4.1

4.3.1. IMPORTING XMI

The creating of UML diagrams is not a part of this research. Therefore, to help users by understanding software design we developed a possibility to import existing 2D UML diagrams.

ArgoUML diagrams can be exported to XMI files. We created an .NET Dynamic Link Library (DLL) to handle the import function. A DLL is used because of the reuse possibility in the visualization software. An example of the XMI file of the UML diagram in figure 4.1 can be seen in appendix A.

Figure 4.1: Sample UML diagram



First, the XMI XML elements are loaded in an internal class structure that describes the different UML entities. We focus only on: classes, interfaces, usages, abstractions, associations, generalization and packages. When a relationship contains stereotypes like “create” or “realize” these are also imported.
The outputs of the DLL method can be used in the 3D engine Unity to create 3D models. The models are loaded in a 3D space and are positioned next to each other. For the prototype, the 3D positioning has to be done by hand to achieve the best readable layout.

4.3.2. 3D UML CLASS DIAGRAM

The software we developed is able to visualize an imported UML class diagram model hierarchy. It is also possible to visualize only a specific part of the model, where the user can select what parts should be visualized. Only UML class diagrams are supported at the moment, but this can be extended with other diagrams.

The UML entities are converted to 3D with the UML drawing conventions in mind. The software draws the connection lines between the entities that represent different associations with the 2D drawing conventions in mind. This means dashed connecting lines in 2D are also drawn in 3D by a dashed 3D connecting line. The arrowheads are also converted according to the UML conventions. A sample can be seen in Figure 4.2

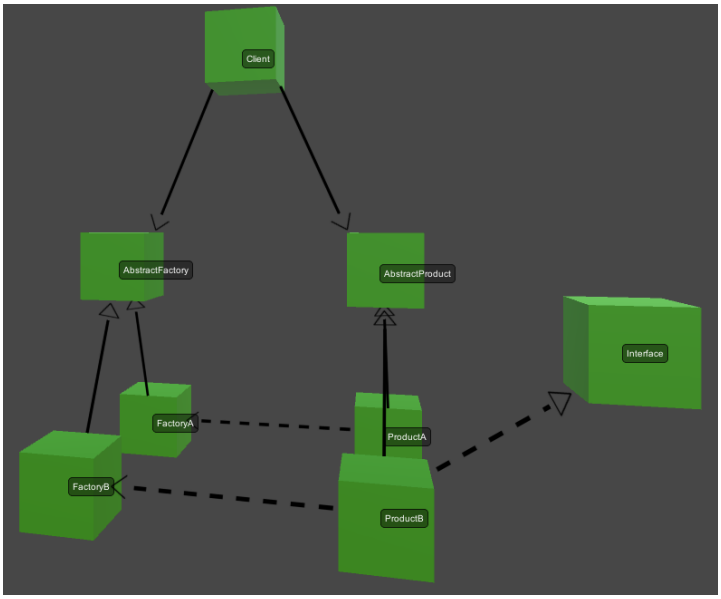
CODE METRICS

With the software tool, code metrics can be visualized by extracting source-code information. The source-code information is displayed with a metaphor.

The metric visualization is based on the existing codecity implementation [17]. The most important metrics used in this implementation are the number of attributes and the number of methods. The prototype software implements the same metrics. The number of methods as height parameter, and the number of attributes as length and width parameter.

Our software uses a simpler version of the codecity metaphor as UML packages are not

Figure 4.2: Sample UML diagram



included in the extraction of metrics. As a results all software classes are extracted to a flattened hierarchy. The software is prepared to support different metrics in the future but this was out of scope for this project.

We have chosen for this metaphor because of the appealing visual view and the relative easy implementation. The basic metrics that we support are good supported by this metaphor. We think that this metaphor enables the user in the best way to quickly see the code-quality aspects without overwhelming the user with visuals.

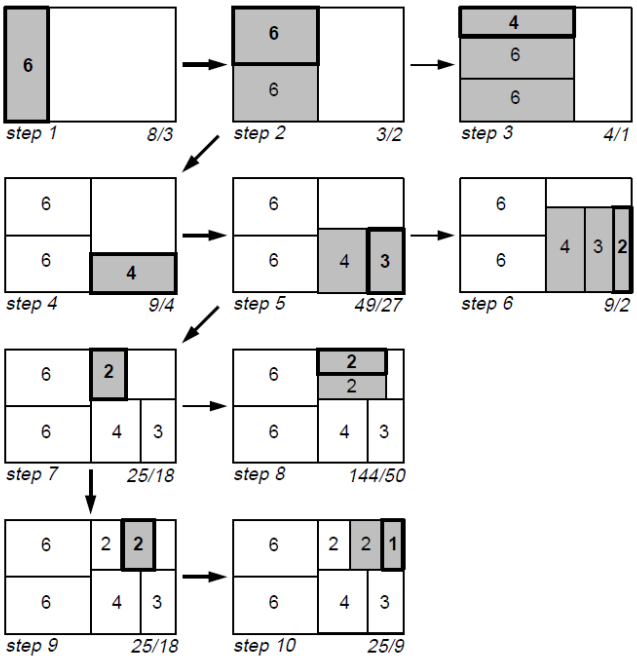
The ordering of the buildings is done with the squarified treemap algorithm that is discussed in the next section.

Squarified Treemaps The buildings in the city metaphor are ordered in a squared treemap pattern. An implementation of an existing treemap algorithm is used [3]. This algorithm calculates the treemap in a way that the space that is reserved for an item in the treemap is always the best fitted square. This makes the presented visualization more readable and better to comprehend. A downside of this squarification technique is that the relative ordering of siblings is lost and images tend to be less regular, with less standard patterns, than standard treemaps. When the structure of the treemap is important and the ordering is not, this method is very useful. A visual explanation of the algorithm can be seen in figure 4.3

In our software tool the ordering of metrics is not important, but the structure. We are interested in the metrics of individual components and not their mutual relationships. This makes it an ideal candidate for the squared treemap implementation.

The squared treemap algorithm is chosen because of the readability of the obtained metrics. The algorithm is created in a DLL and added to the 3D visualization. This made it possible to create 3D codecity implementations from any kind of source-code as long as it is supported by the tool. Figure 4.4 and figure 4.5 show different examples from source-code representations displayed as codecity metaphor.

Figure 4.3: Treemap algorithm example



Implementation The prototype software tool has implemented different ways to extract code metrics from existing projects. The scope is limited to these two import possibilities of source code metric. The prototype tool is able to extend the possibilities in future work.

- Reading and extraction information from .NET DLL and EXE files.
- Reading code metric information from a file created with Eclipse.

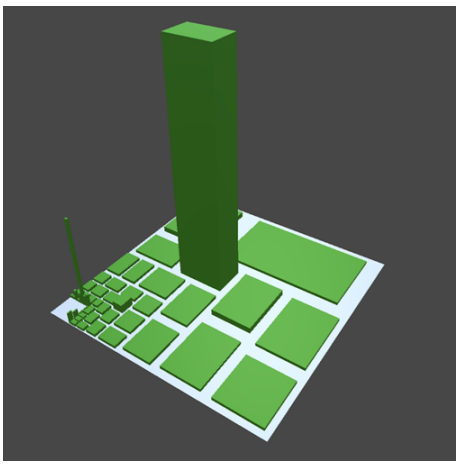


Figure 4.4: Metric example 1

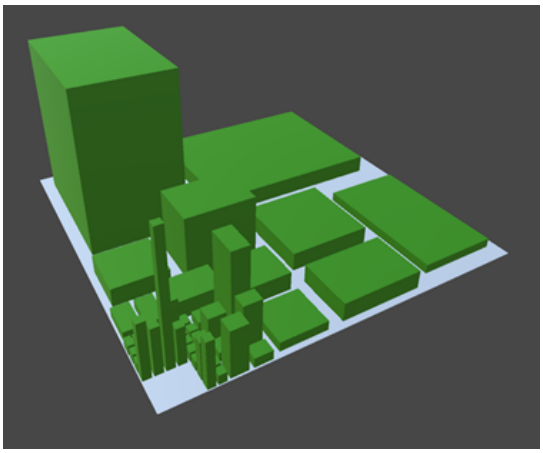


Figure 4.5: Metric example 2

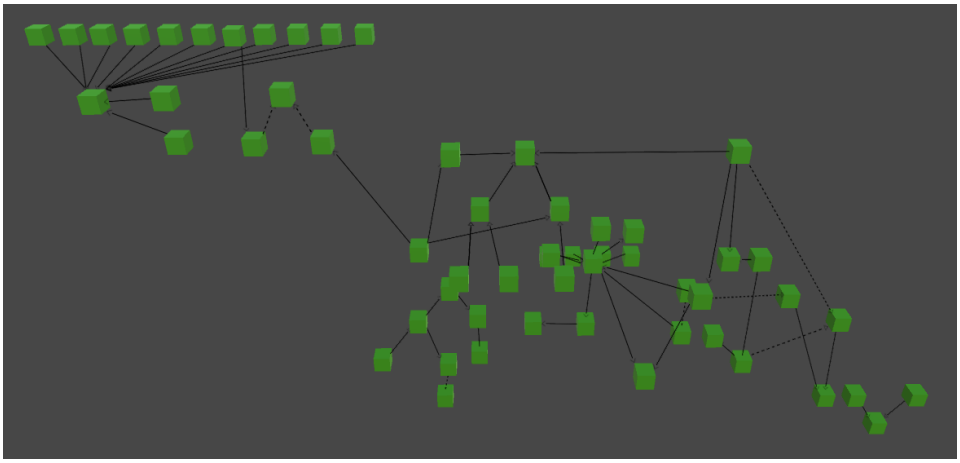
4.3.3. NAVIGATION CONTROLS

The navigation problem is about how the comprehension of the model can be improved by navigation techniques. One of the navigation problems of 2D UML class diagrams is the limitation that often only one diagram can be viewed per window [8]. Constant switching is required to maintain an overview. The software tool we developed has multiple solutions to counter this navigation problem.

First the navigation controls are implemented as used in modern computer aided design (CAD) programs like panning, zooming and rotating. These controls are common in the engineering field, and are widely used in 3D video games. This allows the user to navigate through the complete 3D model without losing focus.

Due to the fact that the 3D diagram can use more drawing space than the 2D variant, the complete model can be viewed without going to the next page that hides a part of the model. 3D navigation makes it also possible to see the 3D model from different viewpoints. Previous work has shown that navigating around the model in 3D adds significantly to understanding the data [16]. In figure 4.6 and figure 4.7 an example can be seen of the same diagram from different viewpoints.

Figure 4.6: Navigation example 1



In figure 4.8 the same diagram is shown but this time zoomed in on a specific part of the model. The navigation possibilities of a 3D UML diagram enables the user to take different viewpoint from the same model without leaving the diagram. We believe the context of the diagram is more preserved this way.

4.3.4. FILTERING

The filtering requirement is implemented to reduce information overload. This is done by providing the user with different filter possibilities. We have developed different filters for different UML entities in the diagram. For example, only “create” stereotype tags or only abstract classes. It is also possible to filter out specific entities. Filtering is all about reducing information overload and reduce cognitive disruption. To remove the back and forth navigation between applications the tool also supports a metric-view that is combined with the UML class diagram. The filtering techniques that are implemented are:

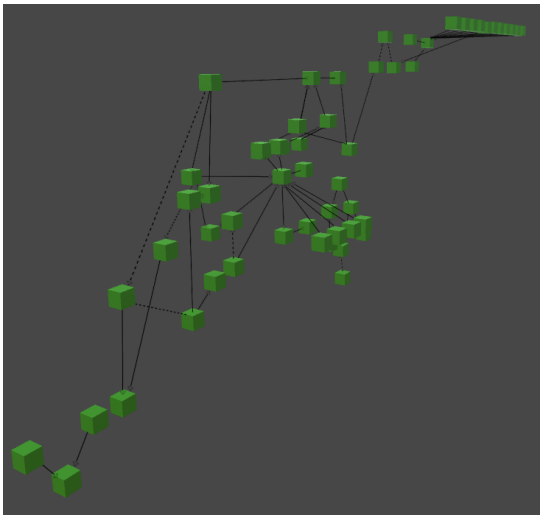


Figure 4.7: Navigation example 2

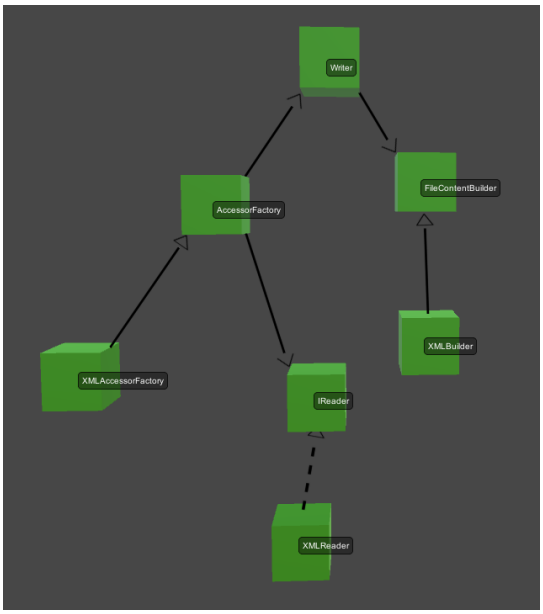


Figure 4.8: Navigation zoomed in

- Filter by name
- Filter by interface or abstraction
- Filter by stereotype
- Filter by selected entity
- Filter by changes in the design and code

It is possible to filter UML entities by name, or a part of the name. The other 3D elements are grayed out. Also the connection lines are made invisible to help finding the entities in a large UML diagram. An example where a filter is applied on the “factory” keyword is seen in figure 4.9

Filter by interface is able to filter the UML entities that are interfaces. This makes it possible to quickly see the interfaces in the UML design. The classes that implement the interface are also highlighted. An example can be seen in figure 4.10

We have also implemented filtering by stereotype. Only a few stereotypes are implemented, but the software is able to extend this to more stereotypes. The software scans all associations for the selected stereotype and the connected UML classes or interfaces. All objects will be hidden excepts of these. This makes it possible to filter certain stereotypes from a larger UML diagram. Figure 4.11 shows an example of a filter on the “create” stereotype.

When an user wants to inspect a specific UML entity, it is possible to filter only on a specific class or interface. The class or interface will be selected along with the corresponding associations and UML entities connected to this association. Only one level of connected entities is visualized. This filter technique is very useful to check if a specific class implements all expected interfaces or check which classes implement an interface. An example

can be seen in figure 4.12 where the “ProductA” class is selected and all connected entities are visible while the other entities are hidden from the view.

The software prototype tool supports the filtering methods as described in this section. This is only a small part what can be done with filtering. The developed software gives an impression what can be possible with a 3D UML representation.

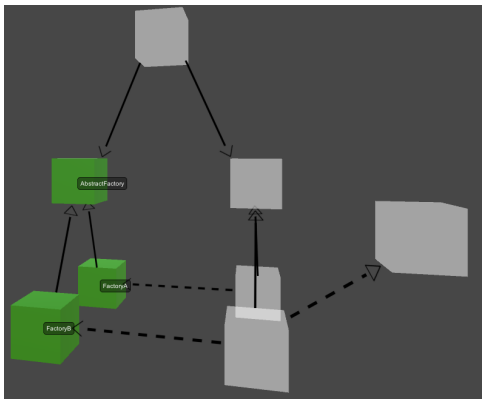


Figure 4.9: Filter by name

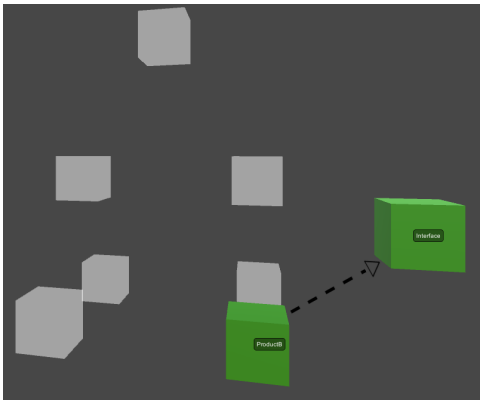


Figure 4.10: Filter by interface

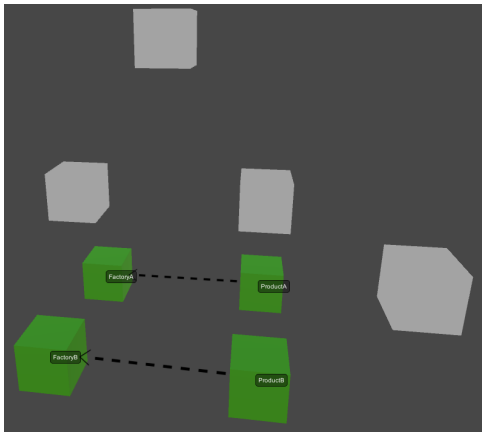


Figure 4.11: Filter by stereotype

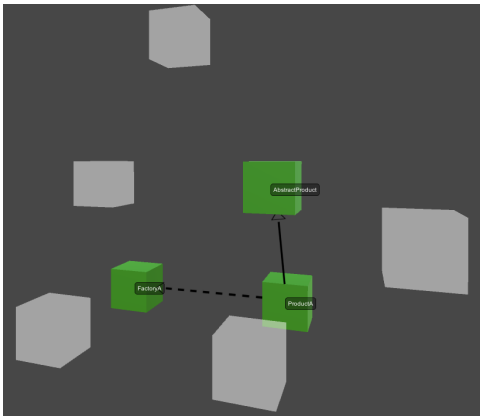


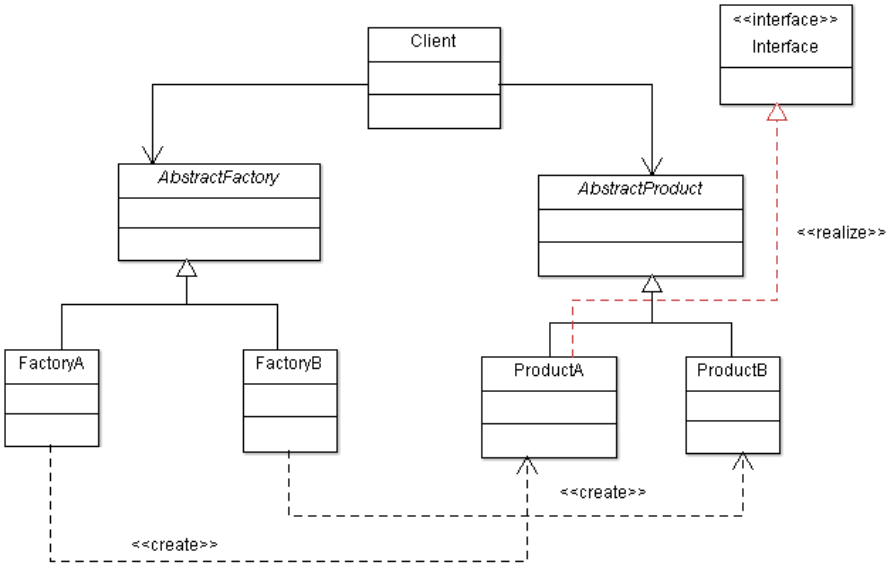
Figure 4.12: Filter by selected entity

VISUALIZE CHANGES

When conducting a change it can be hard for the software engineer to see the impact of the change on the code-quality. One of the possibilities of the software prototype is to visualize changes in the UML diagram. Together with the code metrics visualization, the impact of the design change can be seen. This makes is possible to link a change in the UML design to code-quality. The software is able to compare two different XMI files that contain the same design but with possible changes. For each entity in the UML diagram the changes are shown in the 3D view. This ensures that easily can be seen if and where the UML diagram contains a change. The following differences are possible to visualize in 3D:

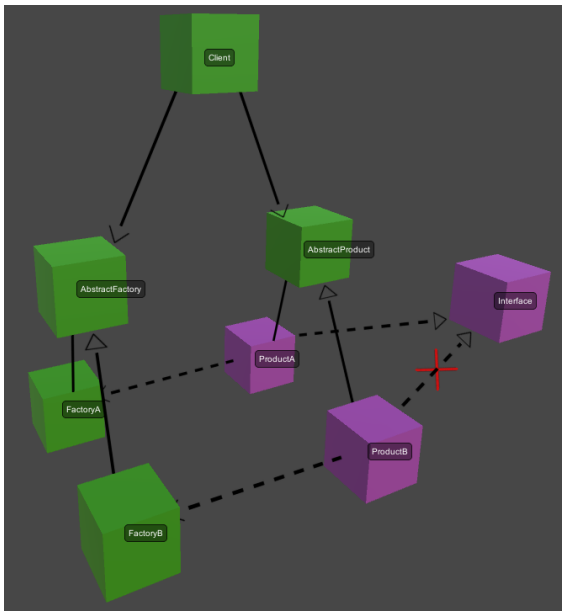
- Missing or added associations
- Missing or added classes or interfaces

Figure 4.13: UML diagram with change



The UML diagram used for comparison is shown in figure 4.13. The UML diagrams are always compared with the design UML diagram as origin. When an association is missing in the UML extracted by the code this is shown by a red cross through this line indicating this line is missing in the code as can be seen in figure 4.14

Figure 4.14: 3D UML with change within 2 versions



The software tool is also able to display changes as text sentences. This gives additional information besides the visual data. This textual data can be used to monitor differences between versions in an automated way.

The comparing algorithm is implemented as a basic algorithm which loops through the UML entities from the initial version and verifies if the compare version contains these

entities. When the same name is found the number of connections with other entities is checked. This way also the differences regarding to the associations are found. Secondly, the compared XMI is scanned for newly added entities. These entities are identified as new items and are visible as such in the 3D diagram and textual representation.

4.3.5. COMBINING UML AND CODE METRICS

When a software design is evaluated both conceptual and architecture information is consulted. The UML class diagram is used for evaluating the (object-oriented) design. The code metrics are used regarding the quality aspects of the source code. This information is not available in one tool, causing for back and forth switching between applications.

To extend the filter possibilities, the software prototype tool has implemented a way to combine code metrics with UML class diagrams. This gives two different viewpoints of the same design. Design and code metrics can be combined to use one technique to examine the other. These two viewpoints are displayed next to each other and are interacting. An example: when a specific class in the UML diagram shows a lot of associations, thus indicating a high coupling problem, this object can be selected. The selected entity in the code metrics view is also highlighted. This makes it possible to directly couple the 3D UML design information to the metrics information and vice verse. When a code metric entity is selected, the corresponding 3D UML entity is also highlighted. An example of a 3D UML diagram and corresponding metric view can be seen in figure 4.15 and 4.16. The green buildings indicate that the corresponding class is found in the 3D UML diagram. The red buildings are classes that are available in the metric information, but not in the UML. This makes it possible to compare different 3D UML diagrams with the same metric information.

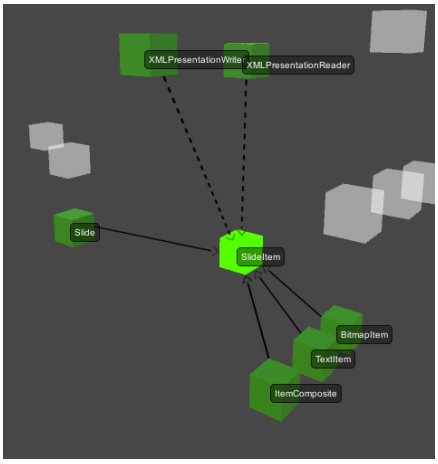


Figure 4.15: 3D UML from example

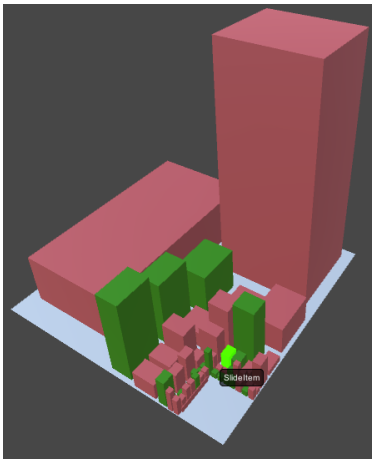


Figure 4.16: Associated metric

5

VALIDATION RESULTS

In this chapter the validation results are explained.

5.1. RESULTS

This sections shows how the problems mentioned in 3.1.1 on page 9 are addressed by the software prototype tool. The validation is done with three different results of students that are handed in as part of the "Design Pattern" course. For each validation aspect a clear example is taken that emphasizes the prototype tool features. The complete results can be seen in appendix B.

5.1.1. FUNCTIONAL REQUIREMENTS

The functional requirements from 3.5 on page 13 are fully implemented. These requirements are necessary to achieve a prototype tool that makes it possible to conduct research on 3D visualization of UML class diagrams.

5.1.2. VISUALIZATION

To help reducing problems concerning visualization as described in 3.1.1 on page 9 we converted 2D UML class diagrams into 3D UML class diagrams. This allowed for more drawing space due to the third dimension. One of the 2D UML class diagram that is used for the evaluation process can be seen in figure 5.1 and the 3D UML variant in figure 5.3.

As we did not expect the extra drawing space does not help with the visual understanding of the UML class diagram. The visual complexity has become even greater in 3D then with the 2D variant. A possible explanation can be that the lines drawn between the UML entities in 3D are a straight line while the 2D lines have corners. The use of cornered lines is easier for the visual understandability. This should be an item for future investigation. Also it depends a lot how the entities are positioned in 3D space. This problem (and solution) is earlier mentioned [5], but is not implemented in the prototype.

Figure 5.1: 2D UML diagram assignment 1-1

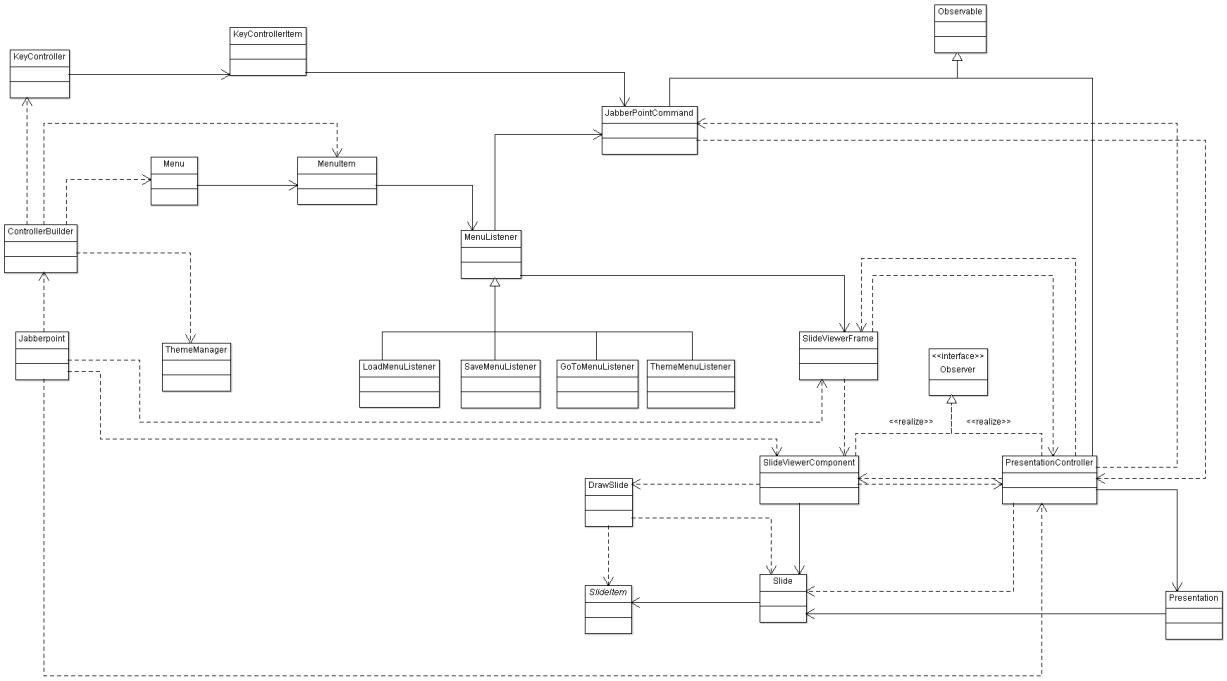
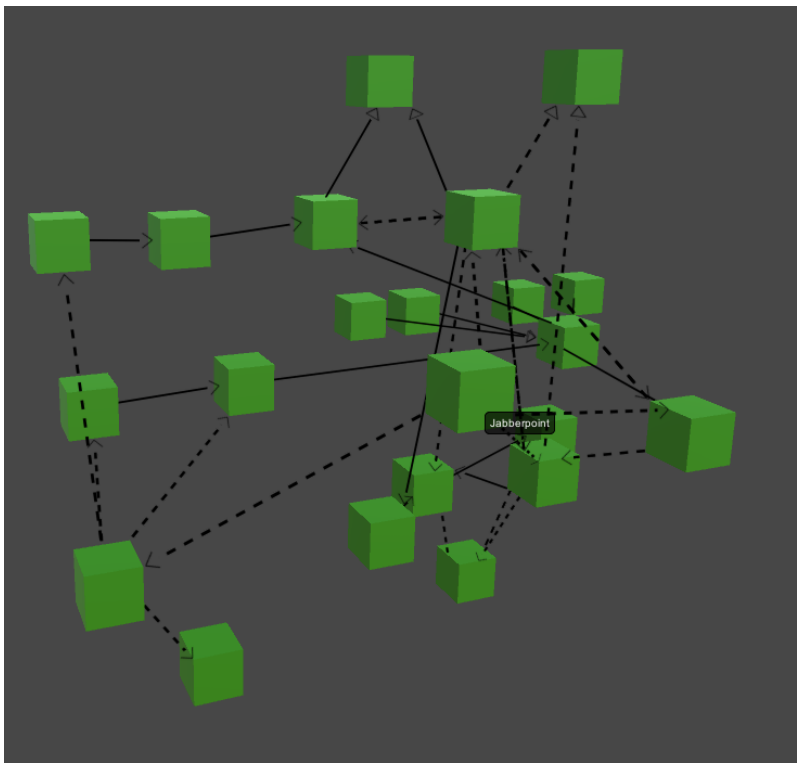


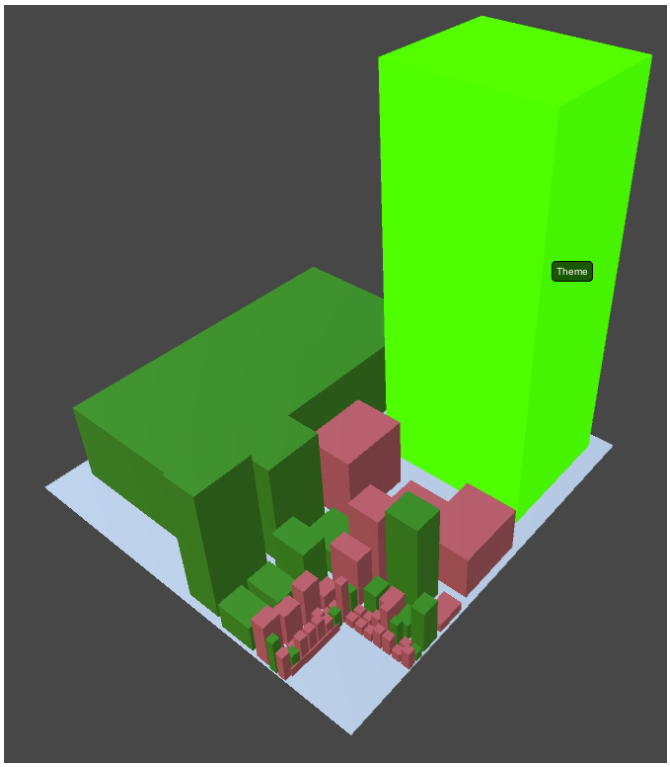
Figure 5.2: 3D UML diagram assignment 1-1



5.1.3. CODE METRICS

The contribution of the metric visualization is validated by examining a result from "Jabberpoint". A 3D metric implemented of the codecity metaphor can be seen in figure 5.3

Figure 5.3: 3D Metric assignment 1-3



It is hard to compare the code metric implementation with the results from the students as the student results do not contain metric information. Only the source-code itself could be used for metrics. We can conclude that the addition of metrics contributes a lot to the understanding of the assignments. It is possible to identify important or large classes without consulting the UML diagram or the source-code. This can help to focus on certain aspects when consulting the UML diagram.

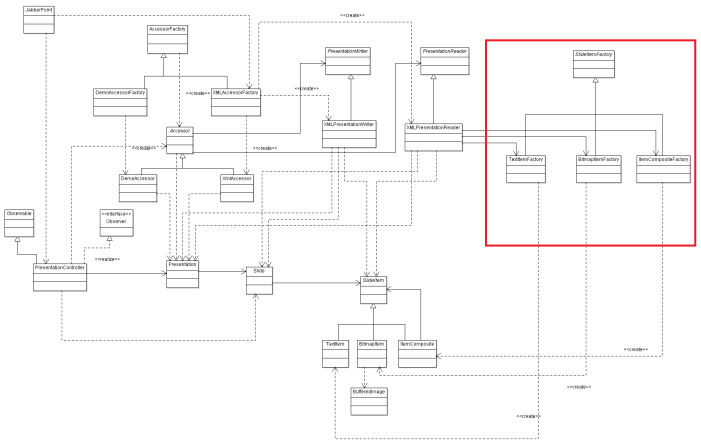
The codecity implementation helps by identifying interesting classes. The fact that multiple metrics can be seen at the same time contributes to the reduction of visual complexity. In one view the important classes can be seen. The metric information is therefore definitely of added value in addition to the UML class diagrams.

5.1.4. NAVIGATION

The added value of the navigation property is validated with the following example:

In figure 5.4 an 2D UML class diagram can be seen. When the user wants to examine the part within the red square the user is not able to avoid crossing lines. When the user has the possibility to view the model from different viewpoints in 3D, this is possible, as can be seen in figure 5.5 and figure 5.6.

Figure 5.4: Validation 3D UML navigation 2D diagram



As can be seen it is possible to navigate through the model to view different contents of the model. It is not necessary to switch between windows. When a viewpoint is not suitable, for example due to undesired line crossings, the viewpoint can be altered without modifying the model. This way for each user the ideal viewpoint can be found.

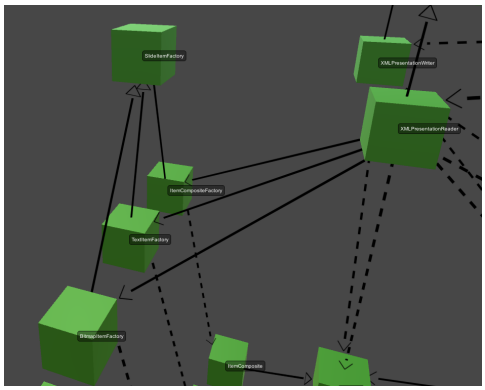


Figure 5.5: Validation 3D UML navigation 1

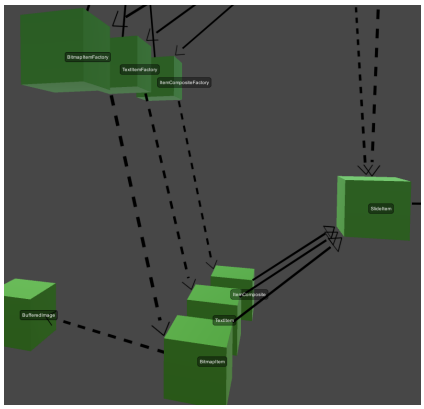


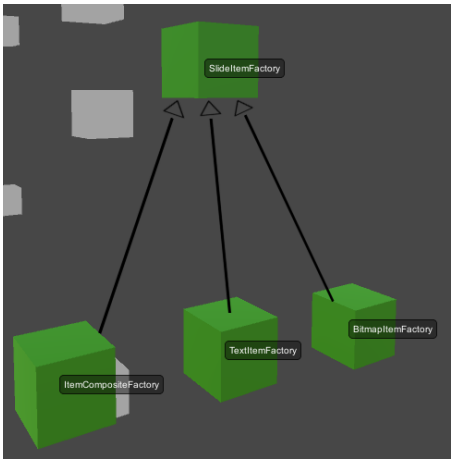
Figure 5.6: Validation 3D UML navigation 2

The navigation controls implemented are the same as in CAD design programs. Also the navigation as implemented is often used in video games. As many software developers are used to these controls this way of navigating through a 3D UML class diagram feels natural.

For the researchers this way of navigating feels natural and therefore helps by understanding the UML model. We find the fact that multiple viewpoints can be taken an added value over traditional 2D UML diagram navigating techniques. One complete diagram can be seen without switching between windows with different models that are connected.

When it is necessary to view details it is convenient to zoom in on specific parts of the model. See also figure 5.7 for a zoomed-in viewpoint of the same model.

Figure 5.7: Validation 3D UML navigation 3

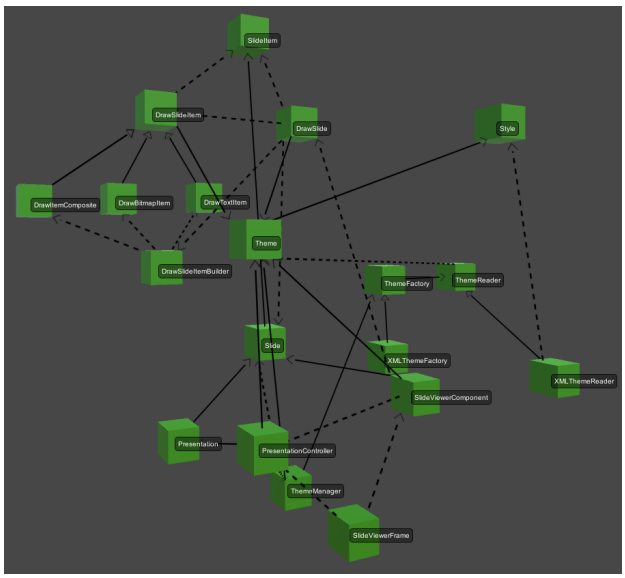


5.1.5. FILTERING

To validate if filtering helps in understanding the UML class diagram, several filter possibilities are implemented. This section presents different filtering techniques implemented in the software prototype tool. These techniques can help the user to understand the diagram in a better way.

The first filter technique is filter by name. The 3D UML visualization in figure 5.8 is used.

Figure 5.8: 3D UML unfiltered



A name-filter with “factory” is applied on the visualization. The result can be seen in figure 5.9. As can be seen it is very clear which part of the diagram contains a “factory”

class. This enables to quickly make a selection in the class diagram and focus on the correct entities.

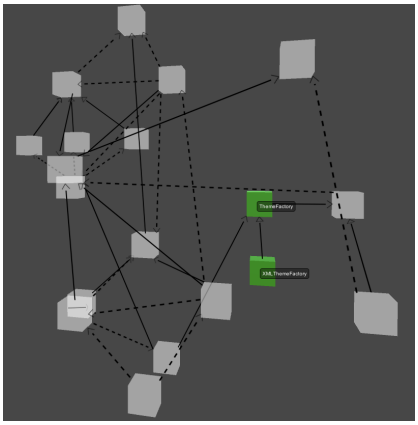


Figure 5.9: Validation 3D UML filtered

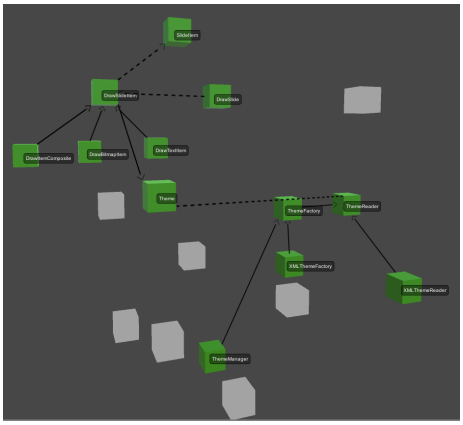
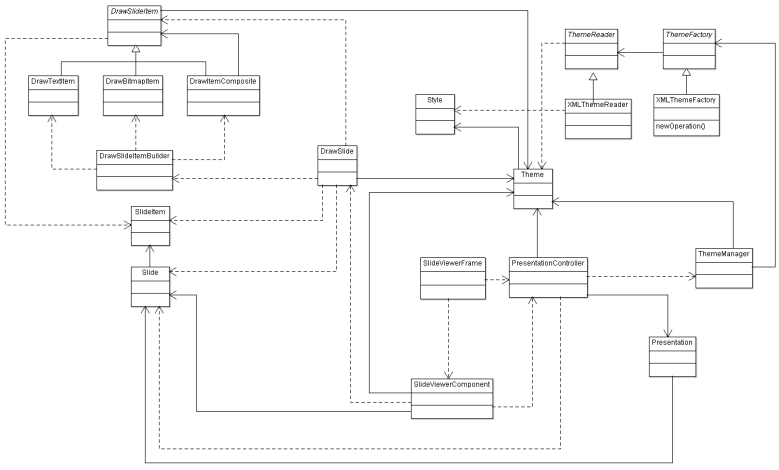


Figure 5.10: Validation 3D UML abstract filter

When searching for the UML entities in a 2D UML class diagram without filter possiblities it is more difficult to find the entities. The 2D UML class diagram in figure 5.11 is used.

Figure 5.11: 2D UML with factory entities



To validate the added value of the “abstract” filter we applied the filter to the visualization. This filter shows only the “abstract” classes and the connected entities. The result can be seen in figure 5.10. As can be seen, with a filter the entities are much easier to find than to manual find the entities in the diagram. Another validation is done with another diagram.

We applied a filter on a “realize” stereotype. Only the connections between entities with a realize stereotype are visible with their connected entities. The unfiltered diagram can be seen in figure 5.12 and the filtered diagram in figure 5.13

The filtering techniques as described help to emphasize key area‘s of the model. The way the filtering is implemented the outline of the complete model is still visible while the

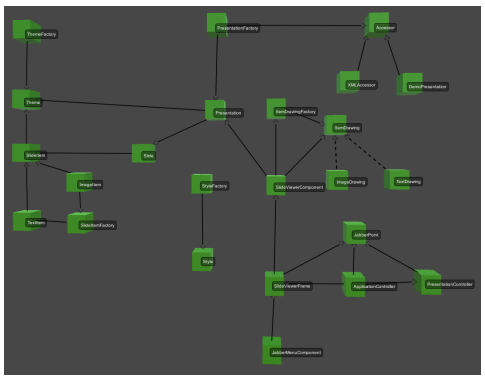


Figure 5.12: Validation 3D UML unfiltered

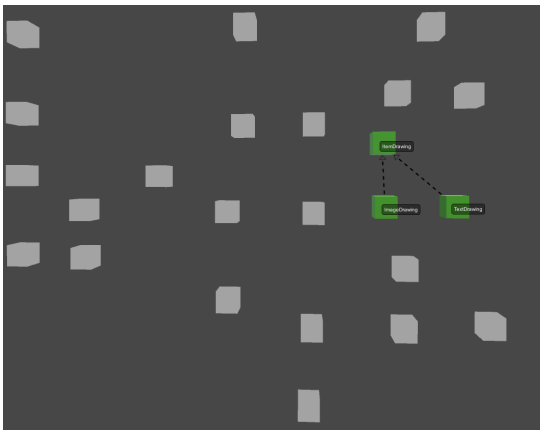


Figure 5.13: Validation 3D UML stereotype filtered

filter is active. This way the user does not lose track when applying a filter. We found it helpful to filter out specific parts of the model.

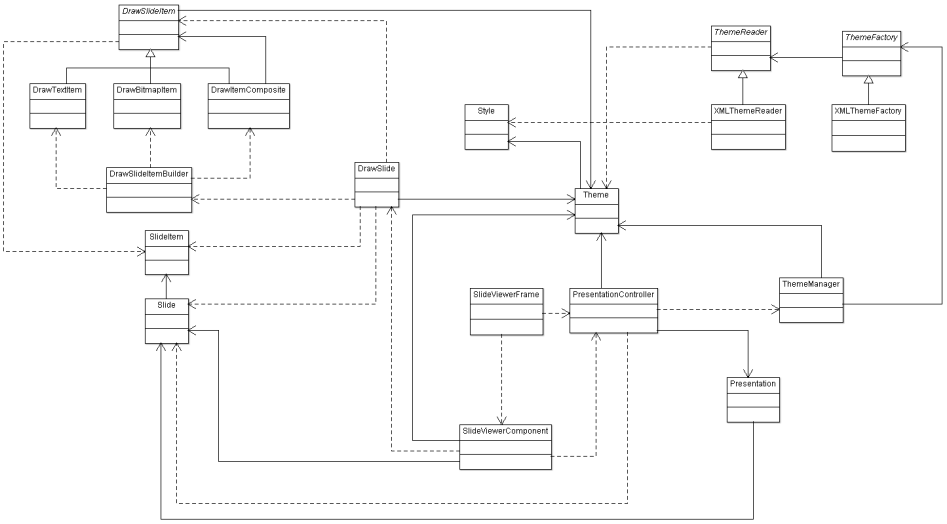
5.1.6. COMBINE METRICS

An extra feature of the filter possibilities is the possibility to combine software metrics with a 3D UML class diagram. To show the added value of this function the following example can be used. In the UML class diagram of figure 5.14 it can be hard to see which UML classes are the most important or at least the classes that require attention. The diagram is converted to a 3D diagram and the metric visualization is displayed in the same view.

The results can be seen in figure 5.15 and 5.16. Now it can be easily seen that the "theme" class requires attention due to the large shape. In one single view it can be seen that the "theme" class contains the most methods of all classes as the height represents the number of methods as described in 4.3.2 on page 17

We find that combining the metric information with the UML class diagram enables to viewer to see "hidden" entities that with *only* the UML class diagram can easily being overlooked. With the metric information, the importance (or size) of certain entities can be identified and the viewer can focus on that specific entity in the class diagram. This saves a lot of time when determining a UML class diagram correctness. For example, in just one mouse-click it can be seen which entities are connected to a selected class in the metric overview.

Figure 5.14: 2D UML unfiltered



These results show that generally speaking, converting from 2D to 3D UML class diagrams without providing extra features does not contribute to the understandability of the diagram. The results show a potential improvement for understanding software design with the filter options providing by the software prototype tooling. Additional with the added metric information these results show that the combination of the two domains help in understanding the software design.

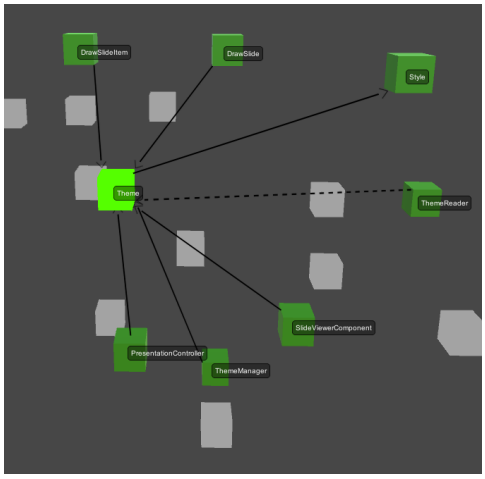


Figure 5.15: Validation 3D UML filtered

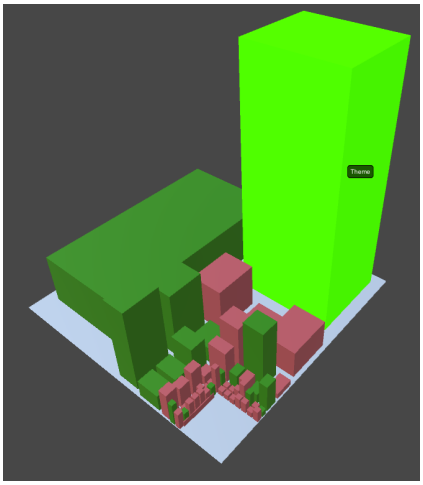


Figure 5.16: Validation 3D UML metric

6

CONCLUSIONS AND FUTURE WORK

This chapter describes the conclusion of this research and gives answer to the research questions.

6.1. CONCLUSIONS

This thesis describes the background, implementation and validation of a software prototype tool that is able to convert 2D UML class diagrams to 3D UML class diagrams. The "XMI" representation of ArgoUML is used. Additionally code metrics are extracted from source code and displayed as the existing "codecity" metaphor. A coupling is made between the UML class diagram and the metrics to help the user understand the quality of the code with the help of metrics.

The overall conclusion is that the software tool is an added value when trying to understand an existing software design. The addition of metrics to the UML class diagram adds to the understandability. Also it has been shown that plain converting 2D to 3D does not have added value as the visual complexity is not reduced.

Now the research questions can be answered to conclude this research project.

How can 3D, compared to 2D, help by understanding software design?

The visualization problem is *not* solved with the developed prototype tool, as has been shown in section 5 on page 25. The 3D visualization alone does not contribute to the understandability. For UML class diagrams the spatial complexity is not reduced when adding a third dimension. Future work should be done on researching alternatives on displaying 3D UML class diagrams that reduces the complexity.

Although the 3D UML class diagrams are not contributing to the understanding of the design it has been shown that the 3D metric information adds a lot to the understanding of the design, and especially the quality of the design. With 3D visualization, more than one metric at the same time can be shown to combine the information given in multiple diagrams in 2D to one view in 3D. This enables more possibilities for displaying code-metrics.

We implemented a “codecity” metaphor to display metrics. There are several more possible metaphors to display metrics, this can be done in future work.

Therefore 3D can help by understanding software design but not by simply converting the 2D UML diagrams to 3D but to add extra information in the form of 3D metrics.

How can navigation controls be implemented in 3D UML class diagrams?

The navigation problem can be solved in 3D by implementing navigation tools borrowed from other fields like computer aided design and video games. These navigation controls are natural for the user and provide no barrier in exploring the model. The navigation options enable the user to take different viewpoint of the 3D model. This way, the user is able to view the model in its own personal best view. Some users might find an overview better to look at, while other users might like a detailed viewpoint for each entity.

Navigation in 3D helps in reducing the number of diagrams. In 2D one diagram can be seen at one time. In 3D it is possible to view larger diagrams. The third dimension is used to break down the model to smaller pieces to focus on. In our prototype tool, it is dependent on the user how the 3D entities are positioned. Therefore the results can vary depending on the diagram.

How can filter possibilities be implemented in 3D UML class diagrams?

Filtering can be implemented in various ways. We have implemented filtering by name, class type, interface and stereotype. This enables the user to quickly make a selection of the view that the user is interested in. When a filter is enabled the entities that are not in the filter criteria are faded out. This enables the user to only focus on the filtered entities. The filter type that we believe will deliver the most added value, is the filter by selected entity. This filter enables only the selected entity and the relations to this entity.

We found in our validation that filtering helps by identifying possible focus points of the software. This mainly came from the metric visualization. With the addition of the “filter by selected” filter possibility is it convenient to identity the corresponding UML entities and therefore the filter possibilities are an useful addition the UML diagrams.

This work leads to the answer to the following research question:

How can source-code metrics be combined with 3D UML class diagrams?

To answer this question we combined all the problem area‘s solutions to display code metrics in collaboration with a 3D UML class diagram. This enables the user to quickly examine an existing software application‘s quality by extraction the source-code metrics. These metrics are displayed in a 3D view and with one single overview the interesting software entities are visible. The user can interact with the 3D metric by clicking on objects that represent software classes or interfaces. The corresponding entity in the 3D UML diagram is selected and the view is filtered on this entity with its connected entities. This enables the user to examine the structural information of a metric diagram and connect

this information to the architectural information of an UML class diagram.

Source-code metric can be combined by linking the source-code metrics to the UML class entities and link the selection between the two views.

6.1.1. MAIN RESEARCH QUESTION

With the answers of the sub questions the main research question can now be answered:

How can we add value to software understanding with 3D techniques?

3D techniques have an added value over 2D techniques. However, only when applied in the correct way. Simply converting 2D UML class diagrams to 3D UML class diagrams does not contribute to the 2D problems from the introduction. When applying different filters on the UML entities, and enable navigation, the investigated problems are mostly tackled. But this cannot be charged to the 3D part. The same techniques could be implemented with a 2D tool and we think that the same results can be achieved. This can be done in future work.

On the other side, with metric visualization, 3D is clearly an added value over 2D metrics. The fact that multiple metrics can be combined in one overview is of added value. This is highlighted by the combination of the metrics and the UML class diagram. The combination of multiple metrics gives a quick, and nice presentable, overview of the software metrics.

The added value of 3D techniques in software understanding can therefore be found in the coupling of 3D metric views to UML class diagrams.

6.2. FUTURE WORK

This sections contains proposals for future work.

6.2.1. 3D UML

We did research the conversion of UML class diagrams. State diagrams are already successfully converted to 3D by others [11]. This suggests that behavioral diagrams are better suited for 3D conversion. Future work can be done on investigating possibilities of transforming other UML diagram types to 3D.

We did transform the 2D UML class diagram to an as close as possible 3D UML class diagram with respect to the UML notation. We want to investigate the possibility to use metaphors in UML diagrams. With software metrics this gives advantages over 2D. This may also be the case for UML diagrams.

The visualization of 3D UML diagrams as implemented is not perfect. Future work can concentrate on improving the visualization by turning the association lines into cornered lines instead of straight. Also experiments should be done with different object representations, as we did only implement one specific representation (a green box).

6.2.2. 2D UML

Our research presented a way to improve the software design with UML class diagrams by implementing features such as filtering and navigation. These features can also be implemented in 2D tools. Filtering features have possible the same positive effects in 2D diagrams. The navigation tools as implemented in 3D cannot be transferred to the 2D space as the third dimension is missing. But extra research can be done on navigation problems in 2D UML diagrams.

6.2.3. METRICS

We did implement one specific metric representation that contains two different metrics. More metric metaphors can be investigated and what metaphor displays information in the best way. Also different metrics should be taken into account, for example “lines of code” and “cyclomatic complexity”.

6.2.4. ADDED VALUE STUDY

The prototype tool should be further developed to a mature tool. This enables to conduct a survey with test-users about the added value. The maturity is needed to ensure the comments of the testers are about the goals of the software and not about the useability.

BIBLIOGRAPHY

[1] T. Ball and S. G. Eick. Software visualization in the large. *Computer*, 29(4):33–43, 1996.

[2] S. Bassil and R. K. Keller. Software visualization tools: survey and analysis. In *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001*, pages 7–17, 2001.

[3] Mark Bruls, Kees Huizing, and Jarke van Wijk. Squarified treemaps. In *In Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42. Press, 1999.

[4] Yael Dubinsky and Orit Hazzan. Using metaphors with software teams. pages 687–690. ACM, 2009.

[5] Tim Dwyer. Three dimensional uml using force directed layout. In *Proceedings of the 2001 Asia-Pacific Symposium on Information Visualisation - Volume 9*, APVis '01, pages 77–85, Darlinghurst, Australia, Australia, 2001. Australian Computer Society, Inc.

[6] Thomas J. Ball and Stephen G. Eick. Software visualization in the large. 29, 04 1999.

[7] I. Jacobson, G. Booch, and J. Rumbaugh. The unified process (reprinted from the unified software development process). *IEEE SOFTWARE*, 16(3):96–96, 1999.

[8] Anne-Katrin Krolovitsch and Linda Nilsson. 3d visualization for model comprehension — a case study conducted at ericsson ab. 01 2009.

[9] A. Kuhn, D. Erni, P. Loretan, and O. Nierstrasz. Software cartography: thematic software visualization with consistent layout. *JOURNAL OF SOFTWARE MAINTENANCE AND EVOLUTION-RESEARCH AND PRACTICE*, 22(3):191–210, 2010.

[10] C.F.J. Lange, Michel Chaudron, and Johan Muskens. In practice: Uml software architecture and design description. 23:40 – 46, 04 2006.

[11] Paul McIntosh, Margaret Hamilton, and Ron van Schyndel. *X3D-UML: 3D UML State Machine Diagrams*, volume 5301, pages 264–279. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[12] J. Nielsen. 2d is better than 3d, 1998.

[13] Juergen Rilling and S. P. Mudur. *3D visualization techniques to support slicing-based program comprehension*, volume 29, pages 311–329. 2005.

[14] Juergen Rilling, Ahmed Seffah, and Christophe Bouthlier. The concept project " applying source code analysis to reduce information complexity of static and dynamic visualization techniques. In *Proceedings of the 1st International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT '02, pages 90–, Washington, DC, USA, 2002. IEEE Computer Society.

[15] M. Termeer, C. F. J. Lange, A. Telea, and M. R. V. Chaudron. Visual exploration of combined architectural and metric information. In *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pages 1–6, 2005.

[16] Colin Ware and Glenn Franck. Viewing a graph in a virtual reality display is three times as good as a 2d diagram. pages 182 – 183, 11 1994.

[17] Richard Wettel and Michele Lanza. Codecity: 3d visualization of large-scale software. pages 921–922. ACM, 2008.

A

XMI FILE EXAMPLE

This appendix shows an example XMI file that is used in the software tool to create a 3D representation.

```
<?xml version = '1.0' encoding = 'UTF-8' ?>
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML'
    timestamp = 'Mon Nov 20 09:59:09 CET 2017'>
  <XMI.header>    <XMI.documentation>
      <XMI.exporter>ArgoUML (using Netbeans XMI Writer version 1.0)</
         XMI.exporter>
      <XMI.exporterVersion>0.34(6) revised on $Date: 2010-01-11 22
         :20:14 +0100 (Mon, 11 Jan 2010) $ </XMI.exporterVersion>
    </XMI.documentation>
    <XMI.metamodel xmi.name="UML" xmi.version="1.4"/></XMI.header>
  <XMI.content>
    <UML:Model xmi.id = '10-75-0--117-445b0ed8:15fc55de477:-8000
       :0000000000000B8C'
      name = 'untitledModel' isSpecification = 'false' isRoot = 'false
         ' isLeaf = 'false'
      isAbstract = 'false'>
      <UML:Namespace.ownedElement>
        <UML:Class xmi.id = '10-75-0--117-445b0ed8:15fc55de477:-8000
           :0000000000000B8D'
          name = 'AbstractFactory' visibility = 'public'
             isSpecification = 'false'
          isRoot = 'false' isLeaf = 'false' isAbstract = 'true'
             isActive = 'false'/>
        <UML:Class xmi.id = '10-75-0--117-445b0ed8:15fc55de477:-8000
           :0000000000000B8E'
          name = 'FactoryA' visibility = 'public' isSpecification = '
             false' isRoot = 'false'
          isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
```

```
    <UML:ModelElement.clientDependency>
      <UML:Usage xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B9D'/>
    </UML:ModelElement.clientDependency>
    <UML:GeneralizableElement.generalization>
      <UML:Generalization xmi.idref = '10-75-0--117-445
          b0ed8:15fc55de477:-8000:0000000000000B91'/>
    </UML:GeneralizableElement.generalization>
  </UML:Class>
  <UML:Class xmi.id = '10-75-0--117-445b0ed8:15fc55de477:-8000
      :0000000000000B8F'
    name = 'FactoryB' visibility = 'public' isSpecification = '
        false' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
    <UML:ModelElement.clientDependency>
      <UML:Usage xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B9C'/>
    </UML:ModelElement.clientDependency>
    <UML:GeneralizableElement.generalization>
      <UML:Generalization xmi.idref = '10-75-0--117-445
          b0ed8:15fc55de477:-8000:0000000000000B92'/>
    </UML:GeneralizableElement.generalization>
  </UML:Class>
  <UML:Generalization xmi.id = '10-75-0--117-445
      b0ed8:15fc55de477:-8000:0000000000000B91'
    isSpecification = 'false'>
    <UML:Generalization.child>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B8E'/>
    </UML:Generalization.child>
    <UML:Generalization.parent>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B8D'/>
    </UML:Generalization.parent>
  </UML:Generalization>
  <UML:Generalization xmi.id = '10-75-0--117-445
      b0ed8:15fc55de477:-8000:0000000000000B92'
    isSpecification = 'false'>
    <UML:Generalization.child>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B8F'/>
    </UML:Generalization.child>
    <UML:Generalization.parent>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B8D'/>
    </UML:Generalization.parent>
```

```
</UML:Generalization>
<UML:Class xmi.id = '10-75-0--117-445b0ed8:15fc55de477:-8000
    :0000000000000B93'
  name = 'AbstractProduct' visibility = 'public'
     isSpecification = 'false'
  isRoot = 'false' isLeaf = 'false' isAbstract = 'true'
     isActive = 'false'/>
<UML:Class xmi.id = '10-75-0--117-445b0ed8:15fc55de477:-8000
    :0000000000000B94'
  name = 'ProductA' visibility = 'public' isSpecification = '
     false' isRoot = 'false'
  isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
  <UML:GeneralizableElement.generalization>
    <UML:Generalization xmi.idref = '10-75-0--117-445
       b0ed8:15fc55de477:-8000:0000000000000B96'/>
  </UML:GeneralizableElement.generalization>
</UML:Class>
<UML:Class xmi.id = '10-75-0--117-445b0ed8:15fc55de477:-8000
    :0000000000000B95'
  name = 'ProductB' visibility = 'public' isSpecification = '
     false' isRoot = 'false'
  isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
  <UML:ModelElement.clientDependency>
    <UML:Abstraction xmi.idref = '10-75-0--117--
       a68c109:15fc95c2949:-8000:0000000000000C75'/>
  </UML:ModelElement.clientDependency>
  <UML:GeneralizableElement.generalization>
    <UML:Generalization xmi.idref = '10-75-0--117-445
       b0ed8:15fc55de477:-8000:0000000000000B97'/>
  </UML:GeneralizableElement.generalization>
</UML:Class>
<UML:Generalization xmi.id = '10-75-0--117-445
    b0ed8:15fc55de477:-8000:0000000000000B96'
  isSpecification = 'false'>
  <UML:Generalization.child>
    <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
       -8000:0000000000000B94'/>
  </UML:Generalization.child>
  <UML:Generalization.parent>
    <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
       -8000:0000000000000B93'/>
  </UML:Generalization.parent>
</UML:Generalization>
<UML:Generalization xmi.id = '10-75-0--117-445
    b0ed8:15fc55de477:-8000:0000000000000B97'
  isSpecification = 'false'>
```

```
    <UML:Generalization.child>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B95'/>
    </UML:Generalization.child>
    <UML:Generalization.parent>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B93'/>
    </UML:Generalization.parent>
  </UML:Generalization>
  <UML:Usage xmi.id = '10-75-0--117-445b0ed8:15fc55de477:-8000
      :0000000000000B9C'
    isSpecification = 'false'>
    <UML:ModelElement.stereotype>
      <UML:Stereotype href = 'http://argouml.org/profiles/uml14/
          default-uml14.xmi#.:000000000000082B'/>
    </UML:ModelElement.stereotype>
    <UML:Dependency.client>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B8F'/>
    </UML:Dependency.client>
    <UML:Dependency.supplier>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B95'/>
    </UML:Dependency.supplier>
  </UML:Usage>
  <UML:Usage xmi.id = '10-75-0--117-445b0ed8:15fc55de477:-8000
      :0000000000000B9D'
    isSpecification = 'false'>
    <UML:ModelElement.stereotype>
      <UML:Stereotype href = 'http://argouml.org/profiles/uml14/
          default-uml14.xmi#.:000000000000082B'/>
    </UML:ModelElement.stereotype>
    <UML:Dependency.client>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B8E'/>
    </UML:Dependency.client>
    <UML:Dependency.supplier>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B94'/>
    </UML:Dependency.supplier>
  </UML:Usage>
  <UML:Class xmi.id = '10-75-0--117-445b0ed8:15fc55de477:-8000
      :0000000000000BA0'
    name = 'Client' visibility = 'public' isSpecification = '
        false' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
```

```
    <UML:ModelElement.clientDependency>
      <UML:Dependency xmi.idref = '10-75-0--116--68
          b9f8e5:15fd8892a25:-8000:0000000000000A6F'/>
    </UML:ModelElement.clientDependency>
    <UML:Namespace.ownedElement>
      <UML:Dependency xmi.id = '10-75-0--116--68
          b9f8e5:15fd8892a25:-8000:0000000000000A6F'
        isSpecification = 'false'>
        <UML:Dependency.client>
          <UML:Class xmi.idref = '10-75-0--117-445
              b0ed8:15fc55de477:-8000:0000000000000BA0'/>
        </UML:Dependency.client>
        <UML:Dependency.supplier>
          <UML:Class xmi.idref = '10-75-0--117-445
              b0ed8:15fc55de477:-8000:0000000000000B8D'/>
        </UML:Dependency.supplier>
      </UML:Dependency>
    </UML:Namespace.ownedElement>
  </UML:Class>
  <UML:Interface xmi.id = '10-75-0--117--a68c109:15fc95c2949:
      -8000:0000000000000C74'
    name = 'Interface' visibility = 'public' isSpecification = '
        false' isRoot = 'false'
    isLeaf = 'false' isAbstract = 'false'/>
  <UML:Abstraction xmi.id = '10-75-0--117--a68c109:15fc95c2949:
      -8000:0000000000000C75'
    isSpecification = 'false'>
    <UML:ModelElement.stereotype>
      <UML:Stereotype href = 'http://argouml.org/profiles/uml14/
          default-uml14.xmi#.:0000000000000834'/>
    </UML:ModelElement.stereotype>
    <UML:Dependency.client>
      <UML:Class xmi.idref = '10-75-0--117-445b0ed8:15fc55de477:
          -8000:0000000000000B95'/>
    </UML:Dependency.client>
    <UML:Dependency.supplier>
      <UML:Interface xmi.idref = '10-75-0--117--
          a68c109:15fc95c2949:-8000:0000000000000C74'/>
    </UML:Dependency.supplier>
  </UML:Abstraction>
  <UML:Association xmi.id = '10-75-0--116--68b9f8e5:15fd8892a25:
      -8000:0000000000000A70'
    name = '' isSpecification = 'false' isRoot = 'false' isLeaf
        = 'false' isAbstract = 'false'>
    <UML:Association.connection>
```

```
      <UML:AssociationEnd xmi.id = '10-75-0--116--68
          b9f8e5:15fd8892a25:-8000:0000000000000A71'
        visibility = 'public' isSpecification = 'false'
          isNavigable = 'false' ordering = 'unordered'
        aggregation = 'none' targetScope = 'instance'
          changeability = 'changeable'>
        <UML:AssociationEnd.participant>
          <UML:Class xmi.idref = '10-75-0--117-445
            b0ed8:15fc55de477:-8000:0000000000000BA0'/>
        </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
      <UML:AssociationEnd xmi.id = '10-75-0--116--68
          b9f8e5:15fd8892a25:-8000:0000000000000A72'
        visibility = 'public' isSpecification = 'false'
          isNavigable = 'true' ordering = 'unordered'
        aggregation = 'none' targetScope = 'instance'
          changeability = 'changeable'>
        <UML:AssociationEnd.participant>
          <UML:Class xmi.idref = '10-75-0--117-445
            b0ed8:15fc55de477:-8000:0000000000000B8D'/>
        </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
    </UML:Association.connection>
  </UML:Association>
  <UML:Association xmi.id = '10-75-0--116--68b9f8e5:15fd8892a25:
      -8000:0000000000000A73'
    name = '' isSpecification = 'false' isRoot = 'false' isLeaf
      = 'false' isAbstract = 'false'>
    <UML:Association.connection>
      <UML:AssociationEnd xmi.id = '10-75-0--116--68
          b9f8e5:15fd8892a25:-8000:0000000000000A74'
        visibility = 'public' isSpecification = 'false'
          isNavigable = 'false' ordering = 'unordered'
        aggregation = 'none' targetScope = 'instance'
          changeability = 'changeable'>
        <UML:AssociationEnd.participant>
          <UML:Class xmi.idref = '10-75-0--117-445
            b0ed8:15fc55de477:-8000:0000000000000BA0'/>
        </UML:AssociationEnd.participant>
      </UML:AssociationEnd>
      <UML:AssociationEnd xmi.id = '10-75-0--116--68
          b9f8e5:15fd8892a25:-8000:0000000000000A75'
        visibility = 'public' isSpecification = 'false'
          isNavigable = 'true' ordering = 'unordered'
        aggregation = 'none' targetScope = 'instance'
          changeability = 'changeable'>
```

```
                <UML:AssociationEnd.participant>
                  <UML:Class xmi.idref = '10-75-0--117-445
                      b0ed8:15fc55de477:-8000:0000000000000B93'/>
                </UML:AssociationEnd.participant>
              </UML:AssociationEnd>
            </UML:Association.connection>
          </UML:Association>
        </UML:Namespace.ownedElement>
      </UML:Model>
    </XMI.content>
</XMI>
```

B

3D UML CONVERSION RESULTS

This appendix shows additional validation results. The conversion of the 2D Jabberpoint results are shown with their 3D conversions.

B.1. JABBERPOINT RESULT 1

B.1.1. DIAGRAM 1

Figure B.1: 2D UML diagram assignment 1-1

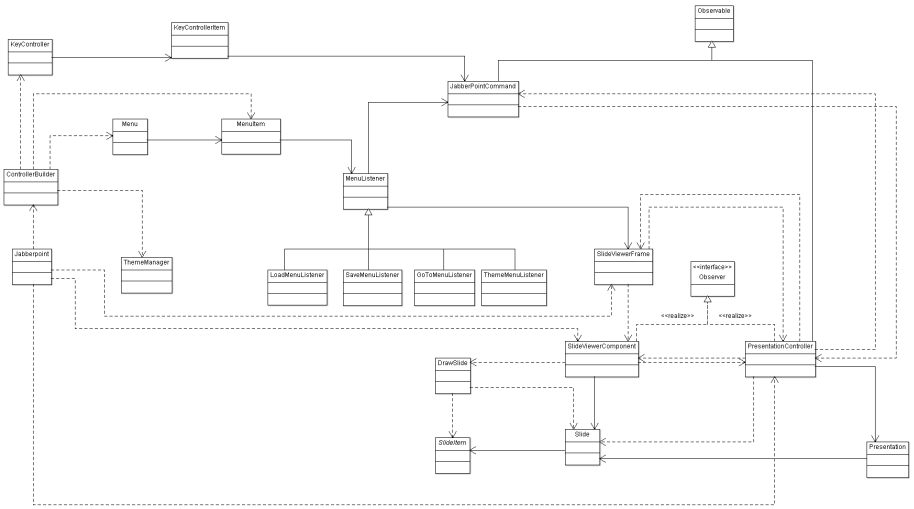


Figure B.2: 3D UML diagram assignment 1-1

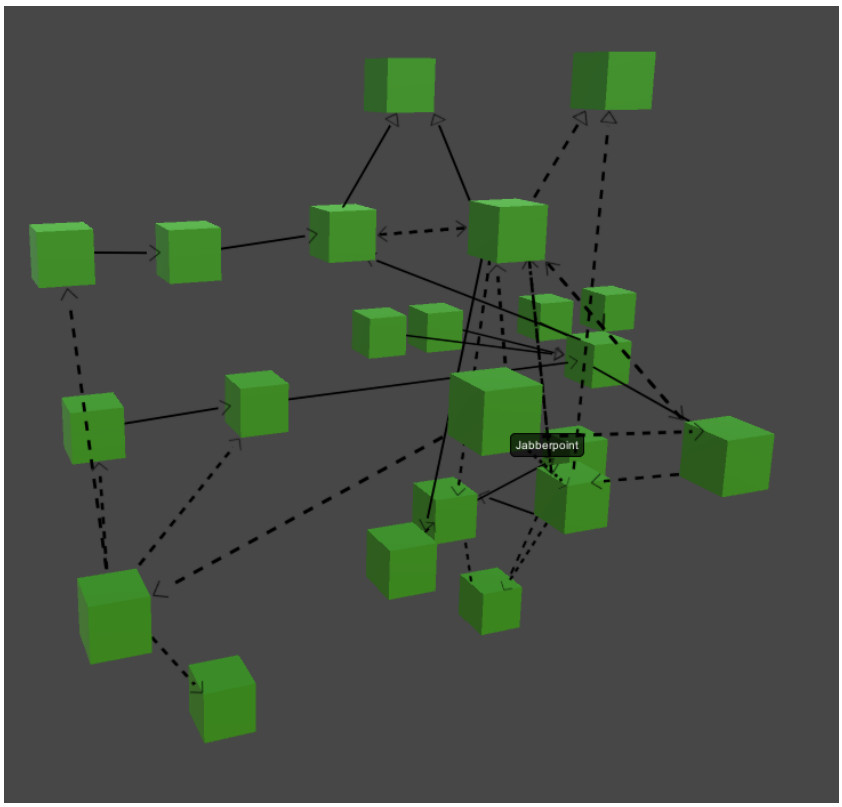


Figure B.3: 3D Metric diagram assignment 1-1

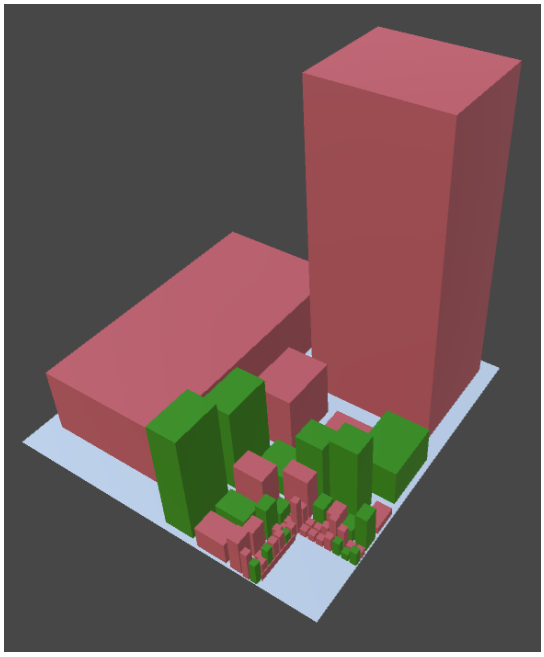
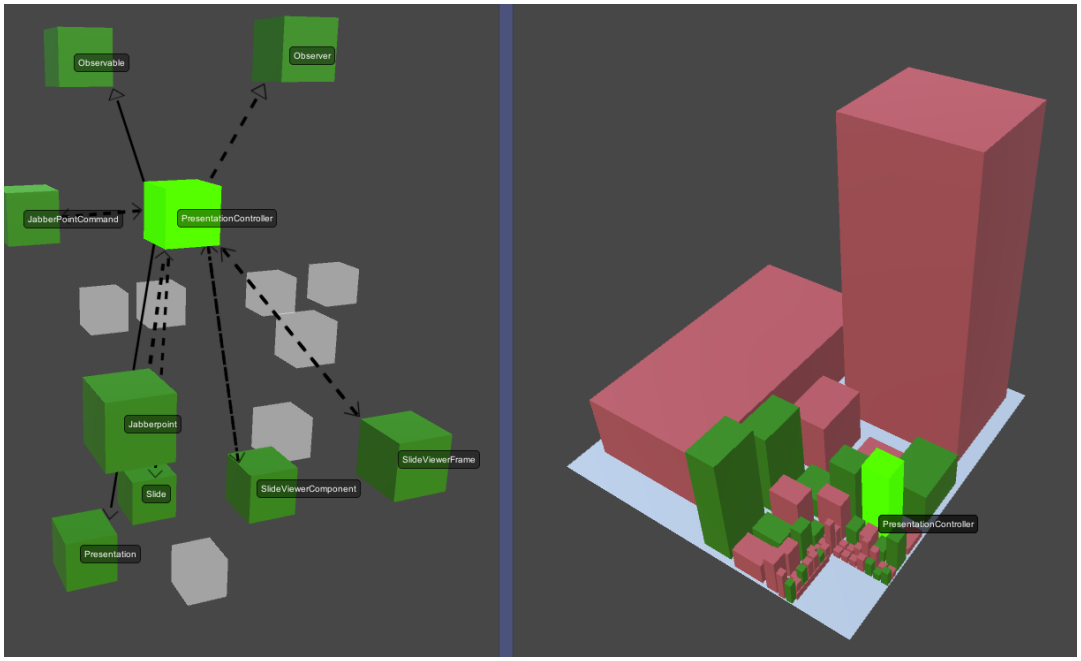


Figure B.4: 3D Metric UML combination assignment 1-1



B.1.2. DIAGRAM 2

Figure B.5: 2D UML diagram assignment 1-2

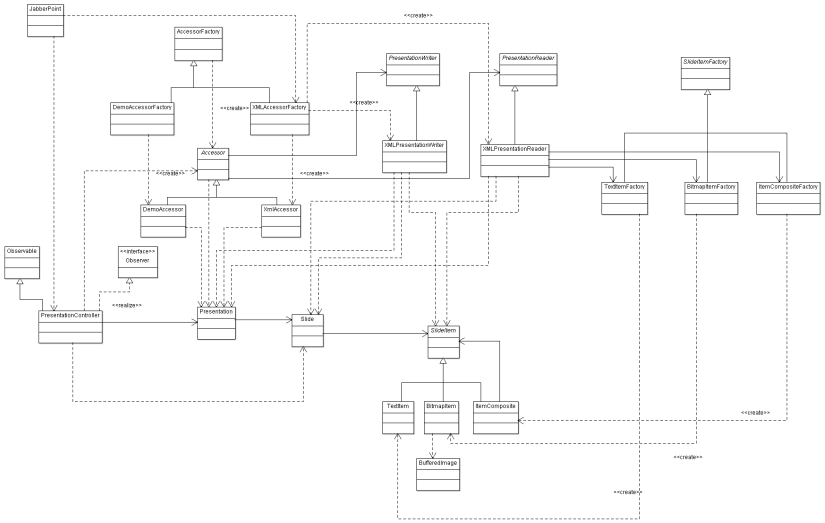


Figure B.6: 3D UML diagram assignment 1-2

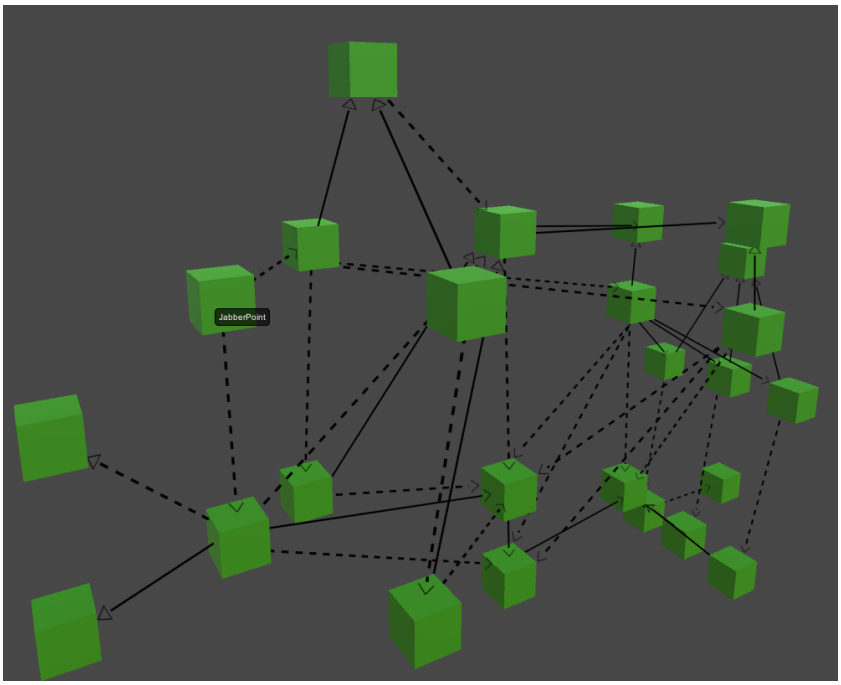


Figure B.7: 3D Metric diagram assignment 1-2

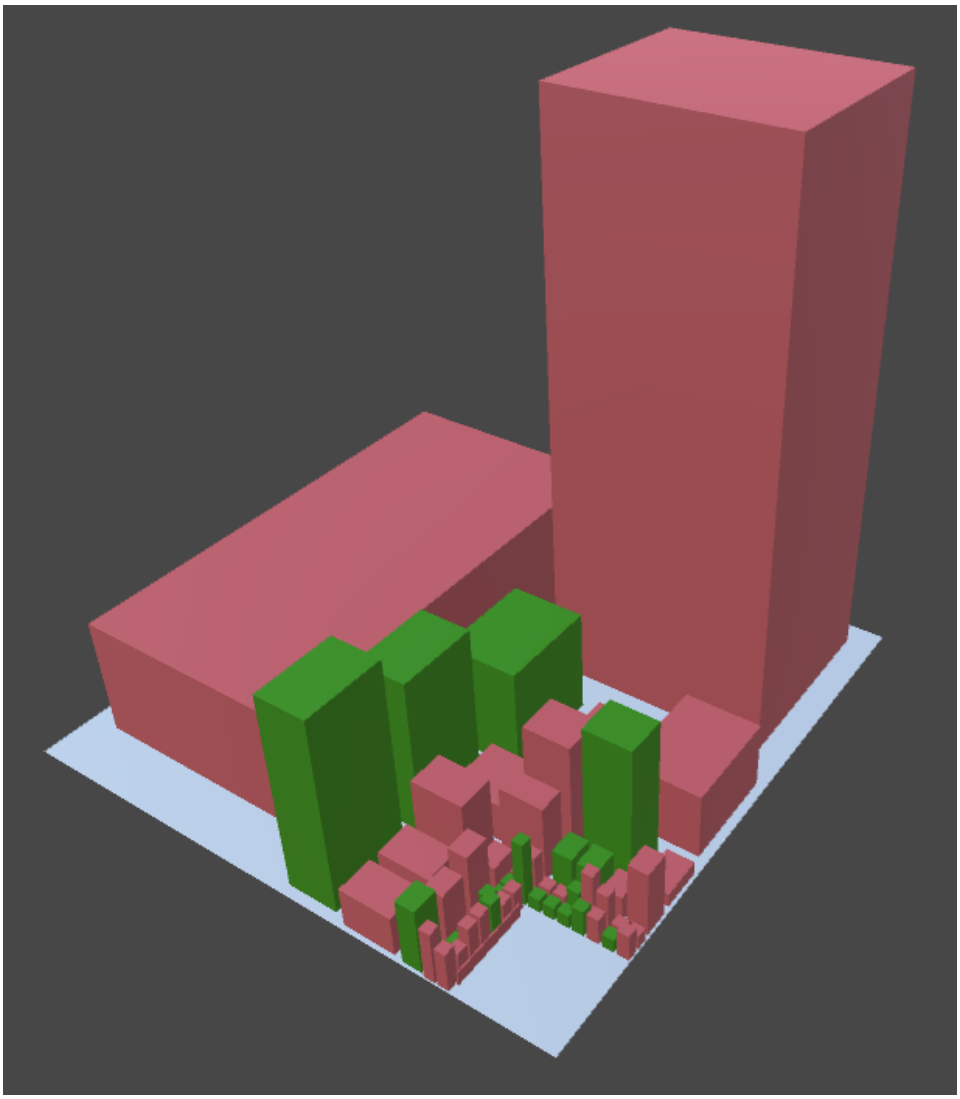
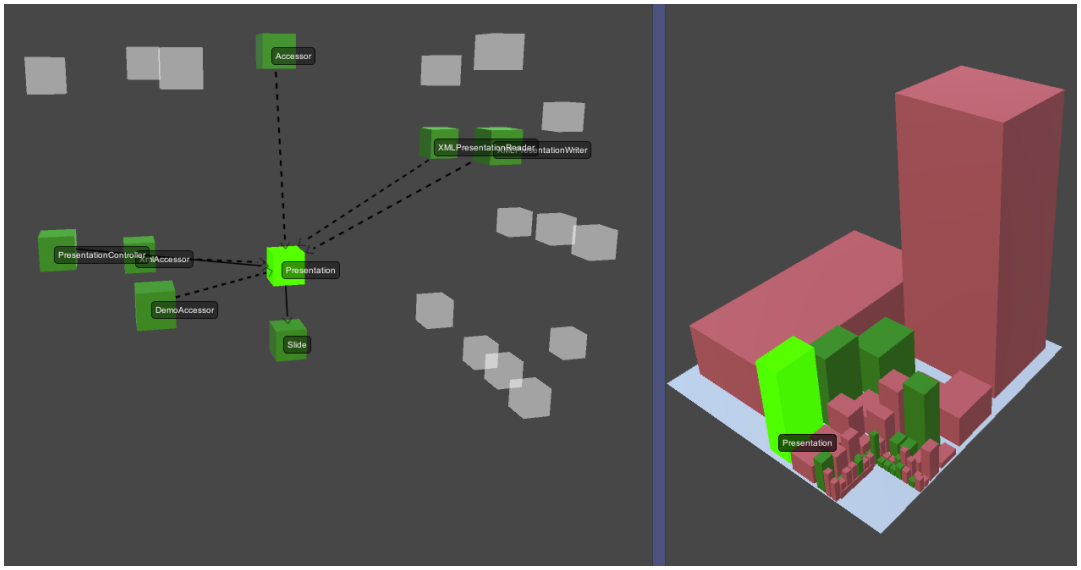


Figure B.8: 3D Metric UML combination assignment 1-2



B.1.3. DIAGRAM 3

Figure B.9: 2D UML diagram assignment 1-3

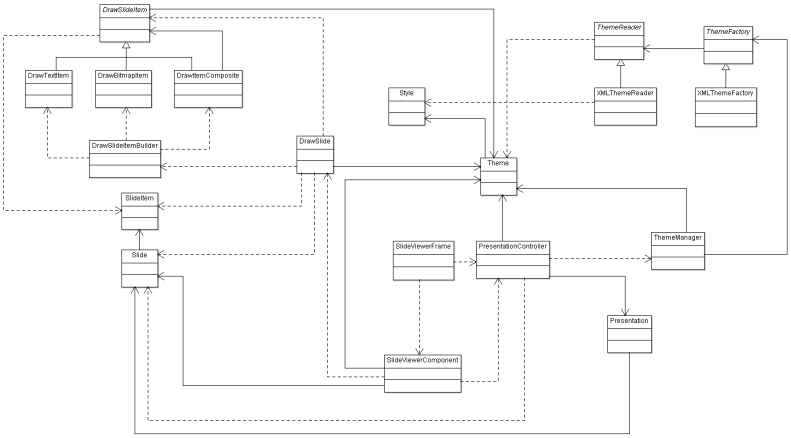


Figure B.10: 3D UML diagram assignment 1-3

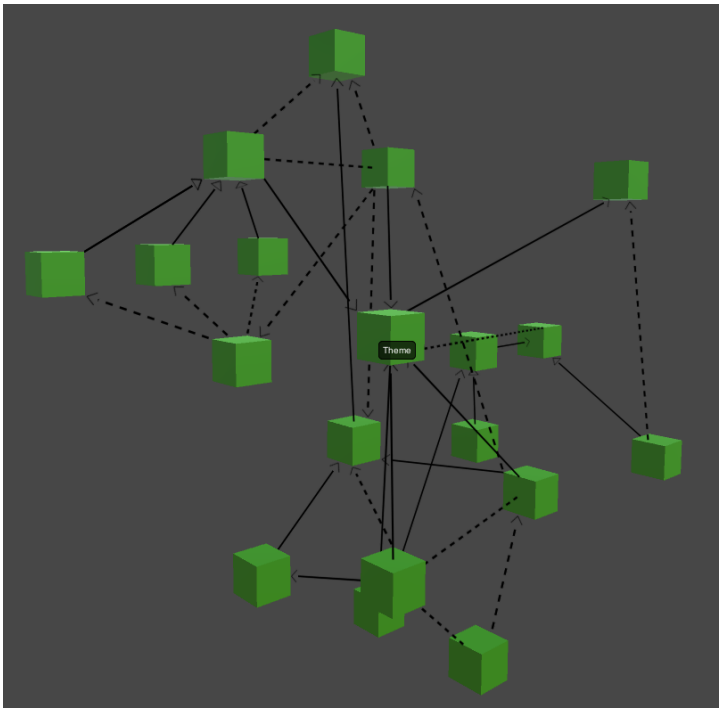


Figure B.11: 3D Metric diagram assignment 1-3

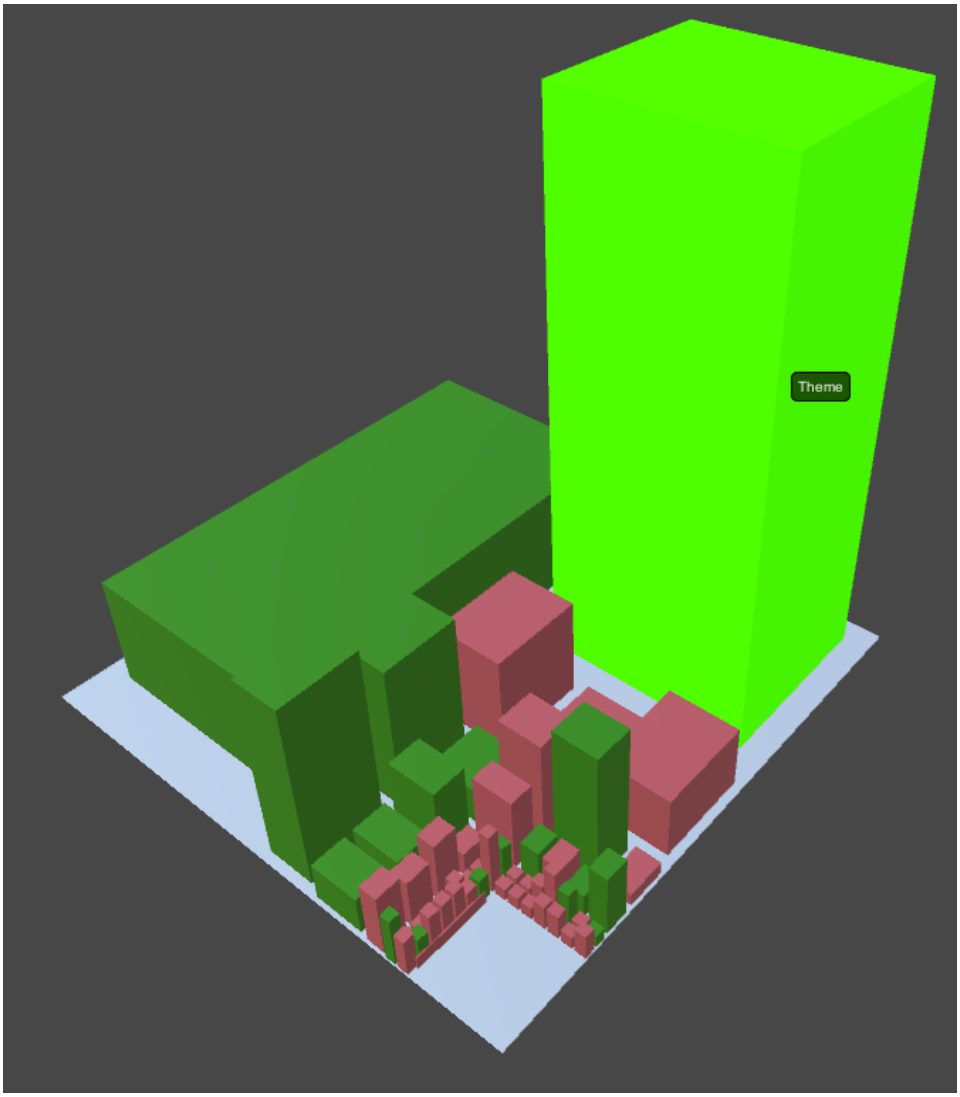
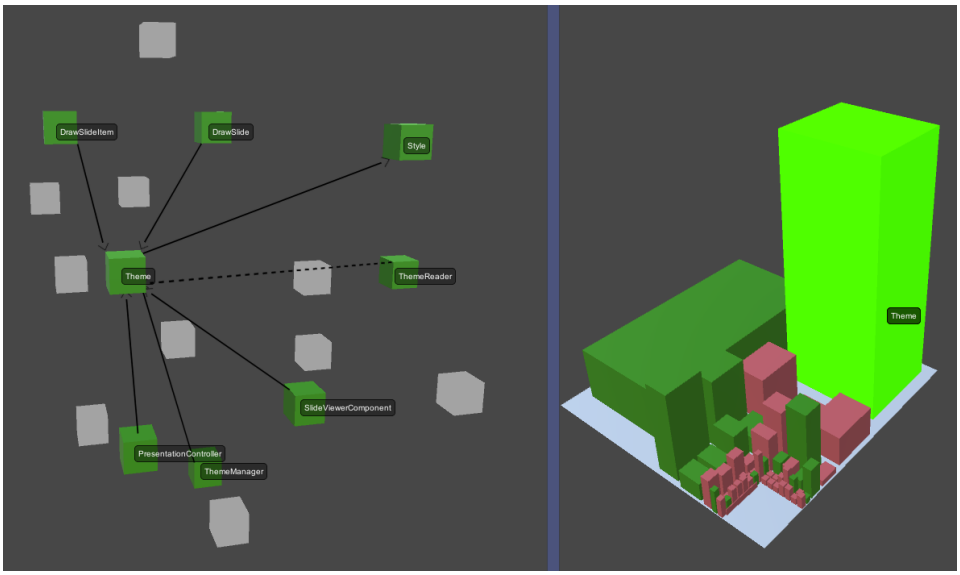


Figure B.12: 3D Metric UML combination assignment 1-3



B.2. JABBERPOINT RESULT 2

Figure B.13: 2D UML diagram assignment 2

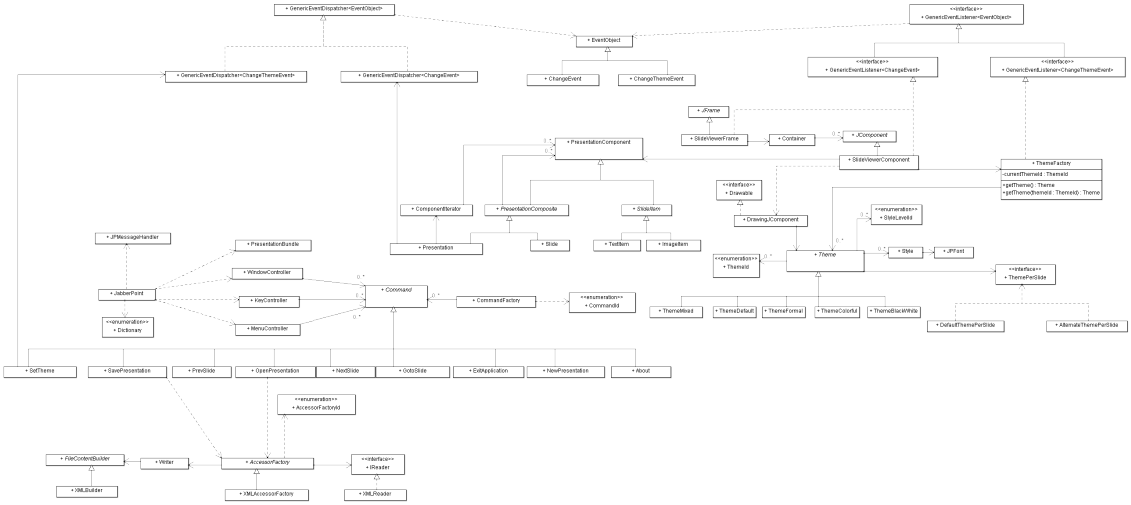


Figure B.14: 3D UML diagram assignment 2

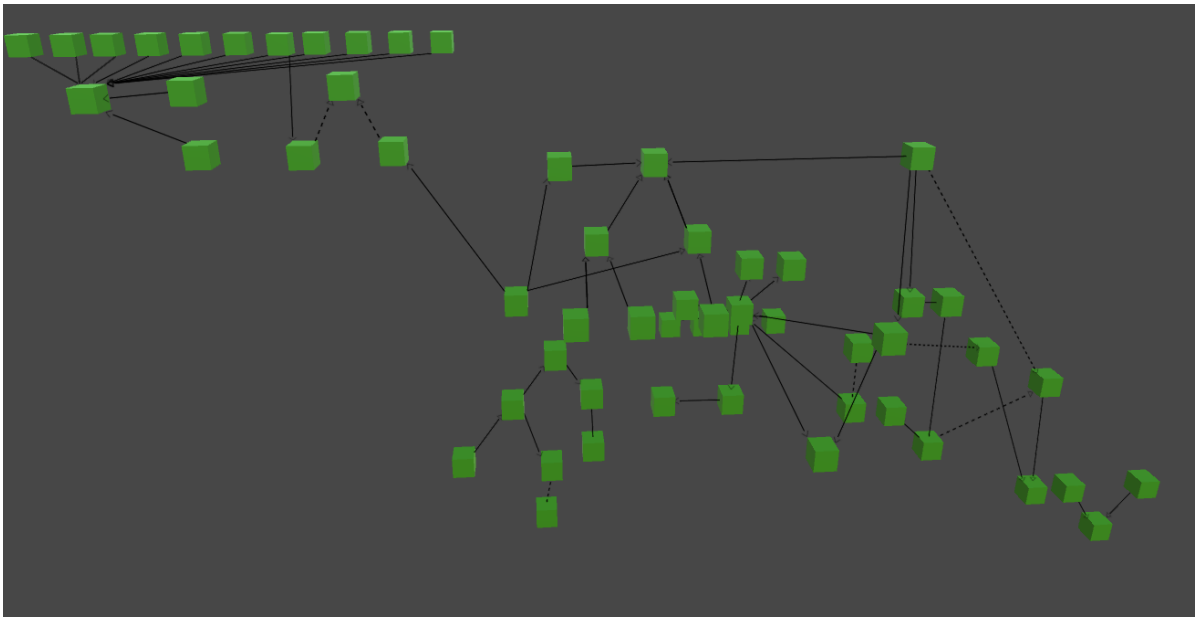


Figure B.15: 3D Metric diagram assignment 2

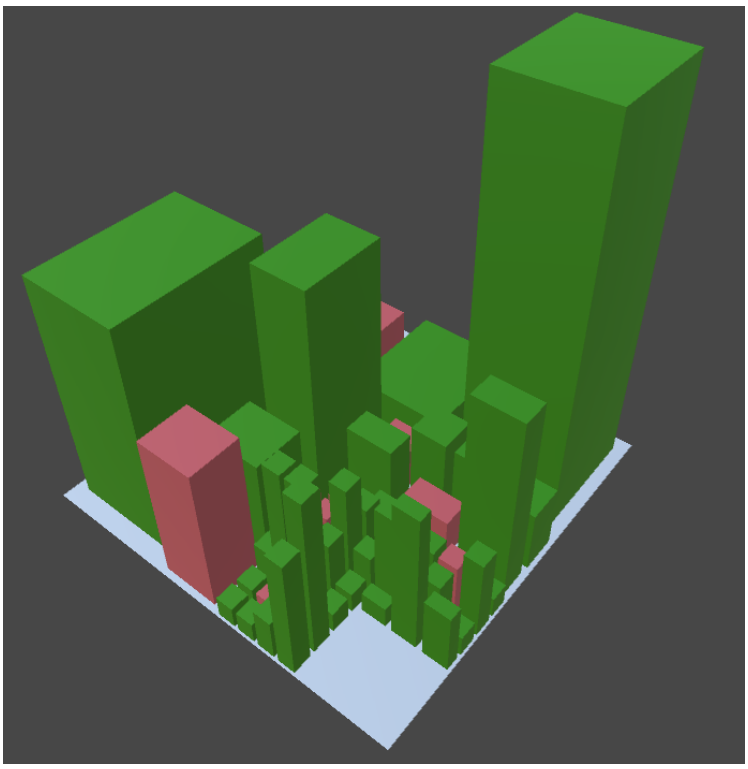
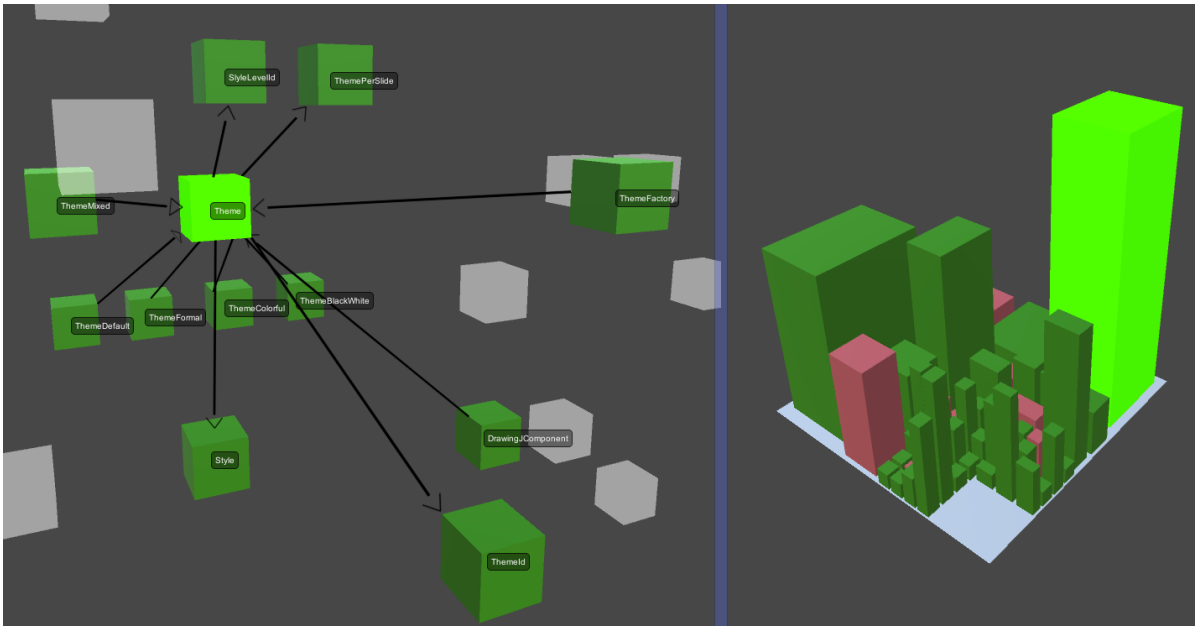


Figure B.16: 3D Metric UML combination assignment 2



B.3. JABBERPOINT RESULT 3

Figure B.17: 2D UML diagram assignment 3

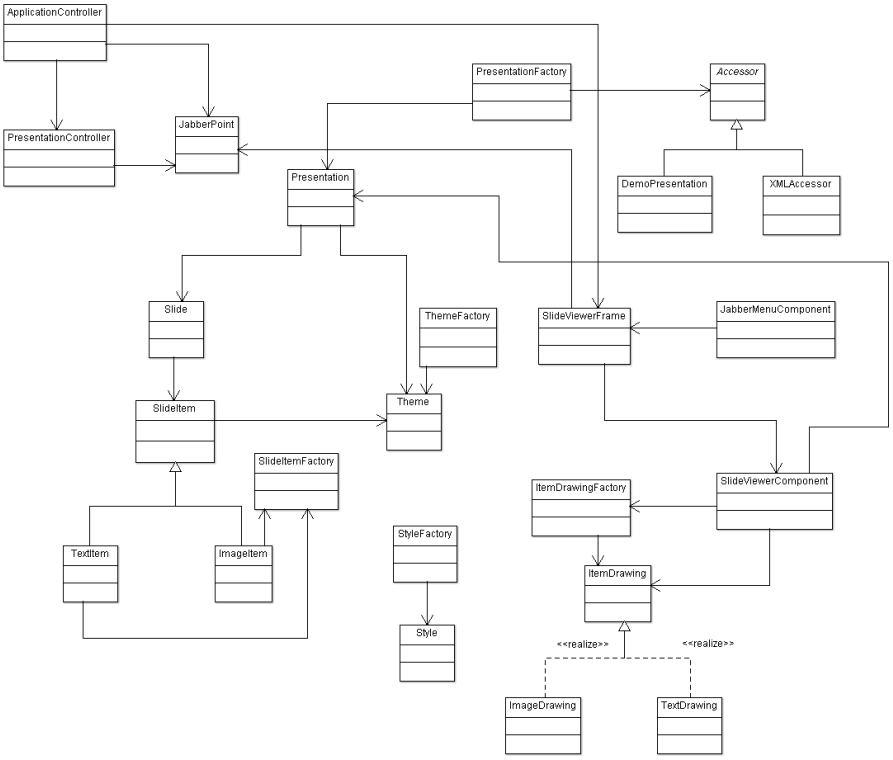


Figure B.18: 3D UML diagram assignment 3

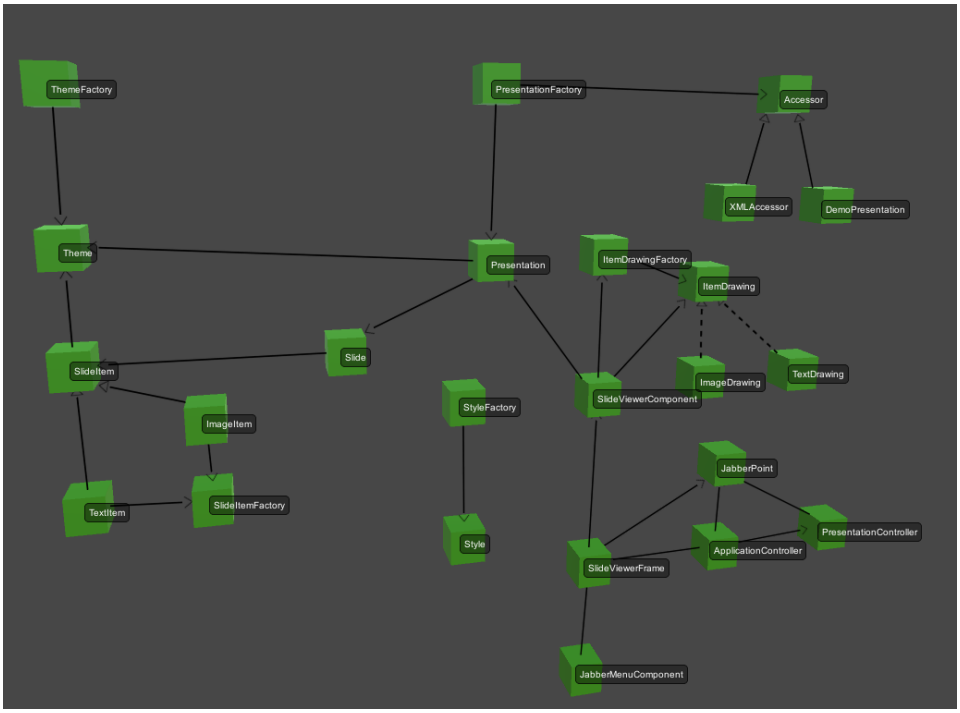


Figure B.19: 3D Metric diagram assignment 3

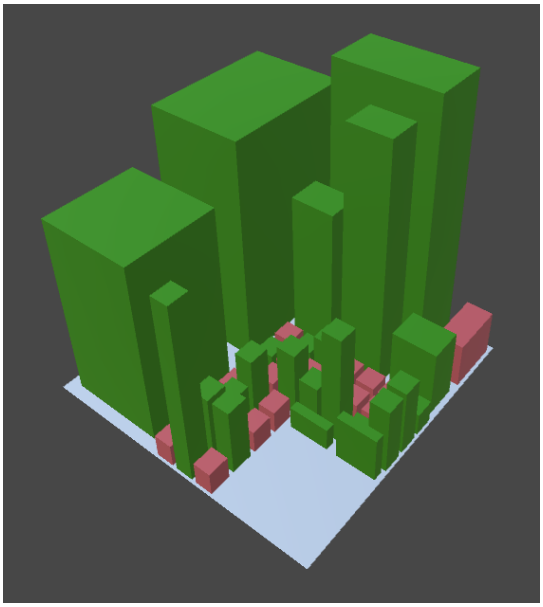


Figure B.20: 3D Metric UML combination assignment 3