MASTERS

User-oriented architectural design
a framework for intelligent guidance through the design process for non-experts

Nimeijer, R.A.

Award date:
2007

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
R.A. Niemeijer
June 20th, 2007

User-oriented architectural design
A framework for intelligent guidance through the design process for non-experts
User-oriented architectural design
A framework for intelligent guidance through the design process for non-experts

R.A. Niemeijer, 513316
Eindhoven, June 20th, 2007

Project website: http://www2.ds.arch.tue.nl/~remco/

Eindhoven University of Technology
Department of Architecture, Building and Planning
Mastertrack Design & Decision Support Systems

Graduation committee:
• Prof. dr. ir. B. de Vries
• Dr. T.A. Arentze
• Ir. A. van der Zee
• Ir. A.J. Jessurun
Preface

This is the final report of my graduation project at the Technical University of Eindhoven. The goal of this project, which will be continued in my PhD project, is to research and develop an interface for a program that allows people to design their own homes.

The subject of giving customers more control over the design of their house has been a recurring one throughout the Master phase of my study. It is a good match for my desire to use Information Technology to improve the relationship and the communication between the building world and the customer. In the second Master project I developed a website for multiple choice houses. The fourth Master project, which was executed as an internship at an architecture bureau, consisted of creating a program which allowed people to make that kind of choices in a 3D environment of the house. They could walk around the house, changing the options in real-time. In this project the scope is again increased: instead of choosing from predefined options people have complete freedom in the design of the house.

In the graduation phase of the project the objective was to research the interface of the program; if people cannot use the software, all other aspects of it are irrelevant. The challenge was to come up with something that could be used by people without extensive computer or design experience.

In order to do this, three rounds of prototypes were created. The first one consisted of paper prototypes, the other two were working programs. These were tested on people with little computer knowledge to see if the designs were usable.

I would like to thank everyone who helped me with this project. First of all the members of the graduation committee: Bauke de Vries for all his help, Theo Arentze for his suggestions and commentary and Joran Jesserun for his help with the UML diagrams and solving a number of problems. I would also like to thank Sjoerd Buma for his assistance in providing me with the materials (laptop, camera, etc.) to conduct the user tests and Jos van Leeuwen, for his help with the first part of the graduation phase.
Summary

The current state of the building industry in the Netherlands has changed little over the centuries when it comes to adapting to the needs of the customer: the architect creates a one-size-fits-all design and any changes the client might want are expensive and time-consuming. Compare this to the automobile industry where changing parts of the car to the customer’s liking is not only possible, but a standard part of the buying process. The material and colour of the seats, the type of wheel hubs, the colour of the car and the engine can all be chosen individually, as can options like air-conditioning, a sun roof or a CD player. In the building industry on the other hand you are lucky if you get to choose between two different kitchen designs.

The problem is not that it isn’t possible: due to technologies like prefabrication and modular constructions it’s no problem at all if not every house is exactly the same. The problem lies with the way most buildings are designed. Most blueprints are still created almost entirely by hand. If a house contains variations, new blueprints will have to be made for all of them, which required manual changes to the original blueprints. Should a change occur in the design after the alternatives have been generated, the change will have to be made in all the variations as well. This is time-costly and error-prone. That is why most contractors only offer a few options, if any, for a house. As the amount of options increases, the time required and the chance of errors increases exponentially.

To solve this problem a change of methodology is required. Blueprints should no longer be created by hand. They should be automatically generated from a 3D model, which can then be (partly) designed by the customer. This saves time (blueprints can be generated in seconds instead of days, if not weeks), reduces errors (unlike 2D drawings, a 3D design can be analysed for mistakes, like rooms that are too small or that don’t contain enough windows) and gives clients far more options to have the design match their wishes.

In order to make this possible, a new system to develop buildings will have to be developed; a system where even clients, who have little to no architectural or technical experience, can design a house. The system will have to check the design for errors and inform the user about them. It should suggest alternatives to give the user inspiration. Designing a house should be no harder than building a LEGO model.

This report describes the process of trying to come up with an interface for the front end of such a system. This was done by creating three prototypes, all with a number of variations for each aspect of the program. These prototypes were then tested on random subjects to determine which of the alternatives was the easiest to use. With this information a set of interface requirements for the final system was determined, which will be used when the system will be developed further in the PhD phase of this project.
# Table of Contents

Preface ................................................................................................................................. 3
Summary ................................................................................................................................ 4
Table of Contents .................................................................................................................... 5

Chapter 1: Introduction ......................................................................................................... 10
  §1.1: Introduction .................................................................................................................. 10
  §1.1.1: Outline ..................................................................................................................... 10
  §1.1.2: Current situation in the Netherlands ..................................................................... 10

§1.2: Project structure ............................................................................................................ 11
  §1.2.1: Graduation phase ................................................................................................... 11
  §1.2.2: PhD phase ............................................................................................................... 11

§1.3: Problem description and goal ...................................................................................... 12
  §1.3.1: Problem description ............................................................................................... 12
  §1.3.2: Goal ........................................................................................................................ 12

§1.4: Thesis structure ........................................................................................................... 12

Chapter 2: Literature Study ................................................................................................ 13
  §2.1: Introduction ............................................................................................................... 13
  §2.2: Related research ........................................................................................................ 13
    §2.2.1: Industrial housing ............................................................................................... 13
    §2.2.2: Mass customization ............................................................................................ 14
    §2.2.3: IFC, OWL ............................................................................................................ 15
    §2.2.4: Easy-to-use CAD software ................................................................................ 16
    §2.2.5: 3D Navigation .................................................................................................... 17
    §2.2.6: Object language and 'intelligent' CAD software ............................................... 18
    §2.2.7: User dwelling design ......................................................................................... 18

§2.3: General principles ...................................................................................................... 20
  §2.3.1: User compatibility ............................................................................................... 21
  §2.3.2: Product compatibility ........................................................................................... 21
  §2.3.3: Task compatibility ............................................................................................... 21
  §2.3.4: Work flow compatibility ....................................................................................... 22
  §2.3.5: Consistency .......................................................................................................... 22
  §2.3.6: Familiarity ............................................................................................................. 22
  §2.3.7: Simplicity .............................................................................................................. 23
  §2.3.8: Direct manipulation ............................................................................................. 23
§2.3.9: Control ................................................................. 24
§2.3.10: WYSIWYG .......................................................... 24
§2.3.11: Flexibility ............................................................. 25
§2.3.12: Responsiveness ...................................................... 25
§2.3.13: Invisible technology ............................................... 25
§2.3.14: Robustness ........................................................... 25
§2.3.15: Protection ............................................................. 26
§2.3.16: Ease of learning and ease of use ................................ 26
§2.4: Usability testing .......................................................... 27
§2.4.1: Definition ............................................................... 27
§2.4.2: The more, the merrier? .......................................... 27
§2.4.3: The Hawthorne effect .............................................. 28
§2.4.4: Usability tests don't measure usability ....................... 28
§2.5: Program analysis .......................................................... 29
§2.5.1: Introduction ............................................................ 29
§2.5.2: Software spectrum .................................................. 29
§2.5.3: 2D design programs ................................................. 30
§2.5.4: 3D design programs ............................................... 31
§2.5.5: Games ................................................................. 33
§2.6: Conclusions ............................................................... 35

Chapter 3: Requirements ..................................................... 36
§3.1: Introduction .............................................................. 36
§3.2: System requirements .................................................. 36
§3.2.1: Non-functional requirements ..................................... 36
§3.2.2: Functional requirements .......................................... 37
§3.3: Use cases ............................................................... 37
§3.3.1: Prototype use cases ............................................... 37
§3.3.2: System use cases ................................................... 37
§3.3.3: Navigation ........................................................... 38
§3.3.4: Products ............................................................. 39
§3.3.5: Object transformations ......................................... 40
§3.3.6: Identifying and correcting problems ......................... 41
§3.3.7: Loading and saving ................................................. 42

Chapter 4: Testing .............................................................. 44
§4.1: Introduction .............................................................. 44
§6.4.4.2: Part 2 - Navigation icons .............................................................. 67
§6.4.4.3: Part 3 - Navigation method ............................................................ 68
§6.4.5: Product selection ................................................................................. 69
§6.4.6: Conclusions ......................................................................................... 70

Chapter 7: Third experiment ........................................................................... 71
§7.1: Introduction ............................................................................................ 71
§7.2: Description of alternatives .................................................................... 71
  §7.2.1: Navigation buttons ............................................................................. 71
  §7.2.2: Working with objects .......................................................................... 72
  §7.2.3: Constraints and warnings ................................................................... 73
  §7.2.4: (Re)placing objects ............................................................................ 73
  §7.2.5: Changing properties ........................................................................... 73
§7.3: Testing ........................................................................................................ 74
  §7.3.1: Method ................................................................................................ 74
  §7.3.2: Design .................................................................................................. 74
  §7.3.3: Use case 1 - Navigation ..................................................................... 75
    §7.3.3.1: Part A - Button layout ................................................................. 75
    §7.3.3.2: Part B - Icons ............................................................................... 75
    §7.3.3.3: Part C - Walking through the house ............................................. 76
  §7.3.4: Use case 2 - Moving and placing objects ......................................... 77
    §7.3.4.1: Part A - Moving a wall ................................................................. 77
    §7.3.4.2: Part B - Selecting and placing a window .................................... 78
    §7.3.4.3: Part C - Moving a window ........................................................... 79
    §7.3.4.4: Part D - Constraints .................................................................... 80
  §7.3.5: Use case 3 - Generic objects ............................................................. 81
    §7.3.5.1: Part A - Changing the properties of a stair .................................. 81
    §7.3.5.2: Part B - Warnings ........................................................................ 82
    §7.3.5.3: Part C - Generic objects vs. products .......................................... 83
    §7.3.5.4: Part D - Replacing a generic object with a product .................... 84
  §7.3.6: Conclusion .......................................................................................... 85

Chapter 8: Conclusions and future research .................................................... 86
§8.1: Introduction ............................................................................................ 86
§8.2: Incremental testing .................................................................................. 86
§8.3: Finalized interface elements .................................................................... 87
  §8.3.1: Viewports ........................................................................................... 88
§8.3.2: Grid .......................................................... 88
§8.3.3: Object transformation .................................................. 88
§8.3.4: Constraints ............................................................ 88
§8.3.5: Navigation ............................................................. 88
§8.3.6: Product view .......................................................... 88
§8.3.7: Property pane ....................................................... 88
§8.3.8: Warnings ............................................................ 88
§8.4: Non-finalized interface elements ...................................... 89
§8.5: Comparison with existing software .................................... 89
  §8.5.1: Viewports ....................................................... 89
  §8.5.2: Grid ............................................................... 89
  §8.5.3: Object transformation ........................................... 89
  §8.5.4: Constraints ....................................................... 90
  §8.5.5: Navigation ....................................................... 90
  §8.5.6: Product view .................................................... 90
  §8.5.7: Property pane .................................................. 90
  §8.5.8: Warnings ....................................................... 90
§8.6: Goal achievement ...................................................... 91
§8.7: Points of improvement ................................................ 91
§8.8: Planning for PhD phase ................................................ 92
§8.9: Planned platform ...................................................... 92
References ........................................................................ 93
Further reading ................................................................... 96
  Books ........................................................................... 96
  Articles and papers ........................................................... 96
Appendix A1 – First round questionnaire ................................... 97
Appendix A2 – Second round questionnaire ................................ 100
Appendix A3 – Third round questionnaire .................................. 103
Appendix B: Final prototype class diagrams ................................ 107
Chapter 1: Introduction

§1.1: Introduction

In this chapter the problem with the Dutch housing industry is described. Since this project will be continued as a PhD study the project’s structure will be described and the problem description and the goal for the graduation phase will be identified. Finally, related research will be indicated.

§1.1.1: Outline

Previous studies\textsuperscript{1,2,3,4} show that if companies want to compete more effectively, they have to meet customers’ needs by offering a large variety of products. In their quest to manage product variety, firms in most industries increasingly are considering product development approaches that can deal with complexity and better leverage investments in product design, manufacturing and marketing\textsuperscript{5}.

Construction firms in countries like Japan\textsuperscript{6}, the USA\textsuperscript{7} and Great Britain\textsuperscript{8,9,10,11} as well as in the Netherlands\textsuperscript{12,13} are starting to see the potential of adopting customer-focused concepts. Increasing competition, a growing demand for variety, the drive for cost efficiency, the growing complexity of construction projects and high inhabitant mobility are some of the forces that make house building firms consider new types of product and process design.

Despite these advantages, the housing industry has been slow to switch to a new way of working compared to other industries. This research aims to develop a system that will make mass customization easier for the building industry.

§1.1.2: Current situation in the Netherlands

The current procedure for a large majority of designing and building in the Netherlands is still based on the principle that the real estate project developers build for the market, without any direct contact with the potential buyer. Users have no input in the design process. At best they are allowed to choose from a few minor alternatives. Changes to the design for individual users are expensive and time-consuming.

The approach of having customers choose from a few options, sometimes called “multiple choice housing”, is becoming increasingly popular. A few examples are covered in §2.2.7: User dwelling design. Others are “Persoonlijk wonen” by Bouwfonds\textsuperscript{14}, 3dClear by Studio Ode\textsuperscript{15} and iBuild\textsuperscript{16} by TNO\textsuperscript{17}. While this approach is still far from ideal, it is at least a step in the right direction.

The amount of alternatives in these projects is always rather limited because the architect has to create separate drawings for each alternative. Having only three sets of three options will result in having to make $3 \times 3 \times 3 = 27$ drawings. This amount grows exponentially as more variations are added; despite most architecture firms having switched to CAD packages like AutoCAD and ArchiCAD, creating drawings has changed little since the pen and paper days. Each drawing is generated by hand, with the operator having to make sure that changes in one drawing are reflected in all the others. Having, for instance, 100 alternatives would quickly become unmanageable.

This problem can only be solved by a fundamental change in the way buildings are designed. Instead of having to draw blueprints by hand, with all the risks of errors that entails, blueprints should be automatically generated from 3D models, which are already created anyway for visualization purposes. This way, buildings can be easily modified without having to worry about keeping all the drawings up-to-date, as they can be automatically recreated in a matter of seconds.
Another benefit of this method, especially in the housing sector, is that because changes are so easy to make, it becomes viable to tailor each house to the needs of the buyer. You could even take the concept further and allow each client to design their own house. Naturally this is easier said then done. Becoming an architect takes many years of study and requires all kinds of knowledge about constructions, aesthetics, government laws, etc. The concept however is intriguing: what if people could design their own houses?

§1.2: Project structure

This graduation project will be continued in a PhD study.

§1.2.1: Graduation phase

In the graduation phase the user interface of the project will be researched, because if the program cannot be used by regular people, the entire application is useless.

The graduation phase will consist of producing a number of prototypes of the interface, which will be tested to see if average people are able to use the program. The structure of the graduation phase can be seen in Figure 1.

The result of this phase will be a set of interface requirements for the final system.

§1.2.2: PhD phase

The PhD phase will focus on actually developing the technology behind the program, such as:

- Generic object and product specification language for building components
- Constraint solving method for interactive manipulation of the 3D geometry, that maintains technical consistency
- AI method for user guidance to support the client in constructing the preferred design
- Design configuration framework that integrates all components and manages the multi-user system

In the PhD phase the focus will be on implementing the backend logic of the program. Prototypes will still be made, but less frequently than in the graduation phase.

Figure 1: Graduation phase work
§1.3: Problem description and goal

§1.3.1: Problem description

The problem is that people today buying a new house don't have nearly enough input into the design of the house. If they can make any decisions at all, they are limited to a handful of pre-designed options.

This is a sharp contrast to the past, where no two houses were alike: every house in the centre of Amsterdam, for example, has a different façade. Only in the last few decades has this concept been abandoned, due to the increasing industrialization of the building industry. While this has decreased costs and errors, it has also led to ever more monotonous neighbourhoods.

§1.3.2: Goal

The goal in the graduation phase of the project is to develop a set of interface requirements for a system that allows people to design their own house. These requirements will be used in the PhD phase of the project to create the interface for the prototype that will be developed.

§1.4: Thesis structure

Chapter 2 describes the results obtained from the literature study. It covers the general principles of user interface design, an explanation of usability testing in general and the method of usability testing that will be used in this project, an analysis of existing software to see how user-oriented design programs currently work and a list of related research.

Chapter 3 details the requirements for the program, from the user perspective and from a technical standpoint.

Chapter 4 describes the chosen testing method.

Chapter 5 describes the creation of the first prototype and the results of the first experiment.

Chapter 6 describes the creation of the second prototype and the results of the second experiment.

Chapter 7 describes the creation of the third prototype and the results of the third experiment.

Chapter 8 draws the conclusions from this phase of the project. A set of interface requirements for the system is defined and that requirement set is compared to existing software. Additionally, an analysis will be done to see which development platform is best suited to implement these requirements and a plan is given for the PhD phase of the project.
Chapter 2: Literature Study

§2.1: Introduction

The first step after identifying a problem is to examine what research has been done on the subject. In this literature study three subjects will be studied:

- Related research
- Literature on user interface design to determine general rules and guidelines that any interface should adhere to.
- Analyzing existing well-known programs related to design to see how the problems of creating an interface for a design program are usually solved

The results of this research will be used in the next chapter, where the program requirements will be defined.

§2.2: Related research

§2.2.1: Industrial housing

In the Netherlands the main focus on industrialized housing occurred after the First and Second World War, due to the large demand for housing at those points. The demand drove the government to seek out ways to build faster and cheaper. Industrialization solved both these problems. This was possible by standardizing building plans and components, so that they could be mass-produced.

However, industrialization can be applied to many different levels of scale. While the floor plans and large components like walls had been industrialized, the assembly was, and still is, largely a manual process. Constructing a house still takes about 600 hours. One reason for this is the high fragmentation in the building industry, i.e. the fact that building is comprised of many different tasks, all of which are done by different professions. The following quotation is from the report of a German committee that was charged with putting forward proposals to save costs in German house building in the 1990s:

"The traditional process of building a simple bathroom begins with the shell (walls and ceiling). Next the heating engineer arrives, followed by the plumber and the electrician (sometimes also the ventilation engineer), both of whom, one after another, install the connections for their own piping and cabling. Next comes the plasterer followed by someone to lay the floor screed. The plumber then returns to install the bath. The next tradesman to come along is the tiler, to tile the floor, the walls and the side of the bath, followed by the joiner, to fit the doors. He in turn is followed by the plasterer to finish the walls and ceiling. The heating engineer, plumber, electrician and ventilation engineer then return in succession to finish their own parts of the job. They are followed by someone to finish off the joints and the grouting. Last of all comes the painter."
If we keep in mind that there is a delay between each of these steps, it is easy to see that a lot of time can be saved by improving the building process.

§2.2.2: Mass customization

The industrialized housing production after the two World Wars came at the cost of variety. Each apartment in a flat was identical, and identical flats were built all over the country. The same occurred with other building types, like row houses. It was not uncommon for a project of 200 houses to have only one house design.

This approach to industrialization is not unique to the building industry: the car industry started out the same way. The original Ford Model T was only available in one colour. Henry Ford is commonly reputed to have said "Any customer can have a car painted any colour that he wants so long as it is black." The association with monotonous housing is part of the reason why industrialization hasn't made any great strides in the building industry.

However, the car industry adapted to offer the customer an increasing amount of choice, while still maintaining the benefits of mass production. The Ford Model T was eventually available in nine body styles, all on the same chassis\(^\text{21}\). When choosing a 2007 Toyota Corolla, you can choose from 3 grades, 2 transmission types, up to five option packages, up to exterior 8 colours, up to 2 interior colours and any combination of 17 accessories\(^\text{22}\). And this is just one of 16 car models in their catalogue. The building industry has not followed suit, despite the market becoming ever more saturated.

When the demand in a market is greater than the supply manufacturers can get away with offering the customer little to no choice. In the last few years however, the amount of new houses built in the Netherlands has rapidly declined.

![Figure 2: Housing supply per year of construction\(^\text{23}\)](image)
As can be seen in Figure 2, the amount of new houses created since 1970 has been growing ever smaller. The building industry has to deal with ‘chain inversion’: the switch from product push to market pull\(^\text{24}\). Customers become ever more critical and will no longer settle for the first available option.

The question then is whether it is possible to give customers more choice without sacrificing the benefits of industrialization. That this is possible can be seen by looking at the clothing industry\(^\text{25}\):

\begin{quote}
The clothing industry has the advantage of being able to mass produce. At the same time short production cycles and the vast assortment of garment available give customers every opportunity to express themselves. MC is made possible by rapidly changing production of many standard designs for stock. Customers select their own ‘clothing components’ and combine them to create millions of different outfits. If different customers would select the same main items, the chance of any one of them coming across someone else wearing the same combination is vanishingly small.
\end{quote}

Because of the saturation of housing market building companies will have to find ways to satisfy the desires of the customers while simultaneously reaping the benefits of industrialization. This concept is called Mass Customization: mass-producing customized housing.

Several studies have been done on the subject of mass customization. One study\(^\text{26}\) has created a discursive grammar in order to be able to structure the demands of the customers. With these demands the computer can create an alternative. The customer can then decide whether the created design is acceptable or if the initial requirements need to be changed, leading to an iterative process of modifying the requirements and judging the result.

Another study\(^\text{27}\) conducted in the UK showed that customers would like increased customization options and that it would be relatively easy for volume house builders to implement. It does note however that the house type approval schemes in the UK might make this more problematic.

A third study\(^\text{28}\) discusses the need to weigh the benefits of mass customization against other factors, such as price, speed of delivery and effort. This study\(^\text{29}\) on the other hand argues for so-called ‘economies of integration’, saying that a better integration of the client in the production process can lead to cost savings, for example by not starting production until an item is actually ordered.

\textbf{§2.2.3: IFC, OWL}

There have been several initiatives to provide a generic building element specification language. The goal of such a language is to allow easy communication between different parties and software applications in the building industry, as they often use proprietary ways of storing information at the moment. By settling on a standard communication protocol data can be exchanged faster and with fewer problems. The two most successful specification languages are IFC and OWL.

IFC (Industry Foundation Classes) is being developed by the International Alliance for Interoperability. It is a standard that’s aimed specifically at the building industry\(^\text{30}\).
In 1995, the IAI was formed to provide interoperability between the software used by all building project participants. The intent is to provide a means of passing a complete, thorough and accurate building data model from the computer application used by one participant to another; with no loss of information. Industry Foundation Classes (IFCs) are data elements that represent the parts of buildings, or elements of the process, and contain the relevant information about those parts. IFCs are used by computer applications to assemble a computer readable model of the facility that contains all the information of the parts and their relationships to be shared among project participants. The project model constitutes an object-oriented database of the information shared among project participants and continues to grow as the project goes through design, construction and operation.

The other standard that’s gaining popularity is OWL (Web Ontology Language), which is also called the semantic web. It’s not specifically aimed at the building industry, but rather it is an XML-like language designed to structure any kind of information.

The Semantic Web is a vision for the future of the Web in which information is given explicit meaning, making it easier for machines to automatically process and integrate information available on the Web. The Semantic Web will build on XML’s ability to define customized tagging schemes and RDF’s flexible approach to representing data. The first level above RDF required for the Semantic Web is an ontology language which can formally describe the meaning of terminology used in Web documents. If machines are expected to perform useful reasoning tasks on these documents, the language must go beyond the basic semantics of RDF Schema. The OWL Use Cases and Requirements Document provides more details on ontologies, motivates the need for a Web Ontology Language in terms of six use cases, and formulates design goals, requirements and objectives for OWL.

There have also been efforts to make the two standards compatible.

§2.2.4: Easy-to-use CAD software

There has been quite a bit of research into CAD software that is easier to use than existing programs like AutoCAD and 3D Studio. However, most of these studies are aimed at architects, and try to create a CAD tool that can be used in the early stages of the design process as an alternative to sketching on paper. Examples include DDoolz34 (Figure 4), which uses cubes as building blocks to roughly sketch out the mass of a design, Cocktail napkin35, which converts 2D sketch input into meaningful objects, Sculptor36 (Figure 3), which uses solids and voids to model a building as if it was made out of clay by adding and subtracting parts of it, Structural Sketcher37, a 2D sketching program, VisTa38 (Figure 5), an urban-scale 3D sketch program and a whole host of others.
Most of the research about navigating in 3D is focused on CAVEs and similar virtual reality systems. A lot of them research input devices like 3D mice, bicycles and treadmills and they often focus on the urban scale. One study researched navigation via paths, where people click on an overhead map to determine the path they will take in the 3D perspective view (Figure 6). This technique is more suited for urban-scale designs than for house scale designs though, due to the small area that is available for navigation and the relatively limited field of view.
§2.2.6: Object language and ‘intelligent’ CAD software

While initiatives like IFC try to describe the physical building components, other studies have focused not on the building components but rather the rooms they create and the functions they represent. In one study, a program called PlaceMaker (Figure 7) was created to study the movement of people through a building. Rooms and objects were entities that have activities related to them. For example a kitchen could be linked to the activity of cooking, followed by walking to the dining room, etc. In order to create the models, a CAAD tool was developed that, like ArchiCAD, is based on parameterized objects. After placing the walls, the user has to define the function of the various rooms so the program can build relationships between the rooms.

Figure 7: PlaceMaker

Because the project is about designing a house using existing building products the research relating to IFC will most likely be the most relevant due to its relatively widespread adoption. Further research in this area will be done at the start of the PhD project, when the language actually has to be implemented in the program.

§2.2.7: User dwelling design

There has been a noticeable trend in Holland towards design that gives the buyer more control. The current effect of that is that quite a few projects offer ‘multiple choice’ houses, i.e. houses where the buyer can choose from a number of pre-designed options, like adding a story onto the house or putting the kitchen at the opposite end of the house. Naturally this has been possible before, but now there are often programs that let the user look at and choose every alternative without the need to consult the architect or project developer, resulting in a lower barrier to do so. These programs tend to be either 2D with pictures of the options or 3D where the user can walk through the building, selecting the various options.
Examples include Woonwijzer (Figure 8), Woonplanner (Figure 9), Woningplanner (Figure 10) and Mini-Makette Online (Figure 11), the latter of which I designed myself.

And of course apart from programs used in practice, research has been done as well. These vary from internet questionnaires that ask a number of questions before automatically creating a house that fits that profile to iBUILD, a project designed by TNO to allow people to customize their houses. While iBUILD is a broad project that looks at the entire building process from product manufacturer to the completion of the building, one part of it is allowing people to design their own house. The system technically allows the user complete freedom, but in practice this freedom is limited to prevent overwhelming the user. Because of this, it has a tendency to resemble the current multiple-choice houses.

The research project most similar to this one is probably MuseV (Figure 12, Figure 13). In it, users can walk or fly around a house and change the design, for example they can move, create and erase walls. This is done on a 30x30 cm grid that only allows orthogonal walls. While this covers the majority of all houses, some design freedom might be lost.

An interesting approach is that rather than treating the physical parts of the building as the core objects, the program uses spaces as the core objects. Walls are automatically generated around a space. A certain amount of AI is included in the program to prevent some illegal decisions.

Another thing that can be noted is that the viewport is separated into a perspective view and a top-down view of the area.
There are many books about user interface design, and most have a number of general principles that any interface should try to follow. Most of these lists share a number of the same items. The one that will be used here is the following:

- User compatibility
- Product compatibility
- Task compatibility
- Work flow compatibility
- Consistency
- Familiarity
- Simplicity
- Direct manipulation
- Control
- WYSIWYG
- Flexibility
- Responsiveness
- Invisible technology
- Robustness
- Protection
- Ease of learning and ease of use

These will be covered in the following paragraphs.
§2.3.1: User compatibility

This is the most important principle of all, as all others more or less flow directly from this one. It can be summarized as “Know the user”. It consists of two basic assumptions: Not all users are the same and not all users are like the developer.

The first one means that it is often incorrect to think of users as a homogeneous group. There will be young and old people using your software, male and female, people with no experience and seasoned veterans, etc., each with his or her own demands on your software.

The second point means that people involved in developing the application are almost automatically unsuited to judging the user interface, because they are too intimately connected with it. Something that might seem perfectly obvious after having used it a couple hundred times for a month might be utterly bewildering for a new user.

Apart from that, the user will generally have a different mindset than the programmer. Programmers are generally more logical than the average user, who tend to be more emotional (consider for example the comment “My computer hates me!”, which a programmer will rarely, if ever, use as he or she knows it to be impossible). Another area where this can be seen is error messages (which will be covered in more detail later on). For example:

"The second day I worked doing phone tech support, I was called by an elderly woman who was sobbing and panicked. After spending twenty minutes getting her calmed down, I finally found out what her problem was. She had been on the Internet and received the ever-popular message "This program has performed an illegal operation and will be shut down." Immediately afterward, she had heard police sirens down the road and thought, "They're coming to lock me up!"

What seems like a perfectly logical error message to one person can drive another to tears.

§2.3.2: Product compatibility

One of the aspects of knowing your user is realizing that they might have experience with similar products or even previous versions of your own product. Radically changing the way things work will therefore cost a lot of time learning the new system. Naturally changes are required, since if there would be no changes there would be no progress, but new ways of doing things must always be weighed against the improvements they bring.

There is a reason that almost all cars, phones, tools, etc. have more or less identical interfaces. People do not want to learn a whole new interface every time they buy a new phone or borrow somebody else’s hammer.

§2.3.3: Task compatibility

This one stems from the difference in mindset between the developer and the user. Users want to get something done, and most of them care very little, if at all, how the computer accomplishes this. They just want to type and print a letter, and whether they do this in Word, WordPerfect or OpenOffice doesn’t matter to them. If they want to include a part of a picture they don’t want to have to open a
separate application: to them the picture is part of the letter and should therefore be handled by the same program.

A good example of this aspect is Microsoft Office. When you insert a picture into Word you see a toolbar that can do simple things like crop, desaturate and lighten the photo so that the user doesn’t have to open something like Paint or Photoshop Elements. You can include graphs without having to open Excel. In PowerPoint you can include video in your presentation so you don’t have to open Windows Media Player separately, etc.

§2.3.4: Work flow compatibility

In her definition of work flow compatibility Deborah Mayhew advocates the ability to multitask. These days however multi-tasking is present in every modern operating system and virtually every program.

Therefore I would like to use a different definition, also stemming from the “Know thy user” principle: know how the user carries out his or her task. Unless the job is computer-related, it has in all likelihood existed dozens, if not hundreds of years before the advent of the computer. It is therefore safe to assume that the user will have a specific way of doing things. This is often ignored in feature specifications, which merely state what functionality the program must have, not how this functionality is typically used.

An example of this would be a hospital order-entry system which, among many other flaws, didn’t conform to hospital workflow. Because of this, nurses kept a separate set of paper records that they entered into the system every night, almost entirely eliminating any benefit the system could’ve brought.

§2.3.5: Consistency

This point is related to product compatibility, but applies within one product as opposed to a range of products. Once the user has learned how to do a task in one part of the program, say deleting an object, the rest of that program should follow that exact method of deleting an object. Consider a command-line interface address book/day planner where the command to remove someone from the address book is “REMOVE [person]” and the command to cancel an appointment is “ERASE [day] [time]”. Infrequent users of the program would always have to look up whether they should use REMOVE or ERASE.

§2.3.6: Familiarity

A computer is a strange and complex thing, and unlike almost anything mankind has seen before. It can do incredibly difficult calculations in milliseconds but doesn’t understand what blue is. It has all kinds of difficult terminology like ‘bit’, ‘cache’ and ‘integer’. Most users (even a lot of computer professionals) don’t know exactly how a computer works, so they make mental models of how it works, based on the knowledge they have of how things work in the real world. These don’t always correspond to reality.

This leads to the following axiom:

A user interface is well designed when the program behaves exactly how the user thought it would.
§2.3.7: Simplicity

Computer programs have a tendency to become more complex over time. More and more features will be added in order to justify new releases or to keep up with the competition. A good example of this is Microsoft Word. Figure 14 shows the rise in complexity (measured by the amount of toolbars and task panels over the different versions of the program. This was the reason for Microsoft to develop a new interface for Word 2007. To quote one of the developers of Word 2007: “I honestly believe we would have had to ship 100 Task Panes in Word 2007.”

![Number of Toolbars and Task Panes in Microsoft Word](image)

*Figure 14: The rise in complexity of Microsoft Word*

This kind of complexity can be overwhelming for many users. When they have to find a function of the program in the midst of dozens of different functions they are likely to give up and write off the program as ‘too difficult to use’.

There are two ways to reduce complexity: one is to have limited functionality in the first place so there’s nothing to overwhelm the user with. This strategy works well for small programs or utilities, but is not a viable option for powerful programs like Word or Adobe Photoshop. The other is to hide the more advanced functions so they do not overwhelm new users but are still available to the experienced users. Google’s search page is a good example of this approach. Novice users can simply specify some terms and click on the search button. Advanced users have a wealth of special syntax available to them to further refine their query.

§2.3.8: Direct manipulation

Partially related to the WYSIWYG concept, this concept states that it is better to allow users to work with objects directly rather than controlling them through something else.

Examples of this are vector drawing programs like CorelDraw and Illustrator (and more and more often bitmap packages too). Rather than resizing a rectangle by say, right-clicking it, choosing Properties, going to the size tab and specifying a new size in inches, only to discover it is slightly too wide and trying values until you get it right, you simply click the object and drag one of the eight handles until it has the size you want.
§2.3.9: Control

Users want to feel in control of their computer, like they do of all their other tools. When they use a hammer, they control it. It will not start hammering away all by itself. When they’re driving a car, they tell it where to go (although this is becoming increasingly untrue with navigation computers, parking help and similar systems). The computer however seems to have a mind of its own sometimes. It will throw up messages like “Could not send your message. Error 421.”, with nothing more than an Ok button. What if it’s not? The message leaves the user with no clue what went wrong, or how to fix it. The famous Windows “Blue Screen of Death” (which fortunately is seen only rarely, if ever, in modern versions of Windows) is another well-known offender of this:

```
A fatal exception 0E has occurred at 0028:C0011E36 in VXD VMM(01) + 00010E36. The current application will be terminated.
* Press any key to terminate the current application.
* Press CTRL+ALT+DEL again to restart your computer. You will lose any unsaved information in all applications.
Press any key to continue.
```

Aside from giving the user information that is completely useless to all but an elite few in the Windows coding team, it also doesn’t provide the user with any control. If you just happened to have an unsaved document in that application, there is no way to save those changes. The user has no choice but to close the program or lose all his or her other work as well.

§2.3.10: WYSIWYG

WYSIWYG is an acronym for What You See Is What You Get. It originated in the time when command-line interfaces like DOS were being replaced with graphical interfaces like the Macintosh, which unlike their predecessors had the ability to actually show what a document would look like before it was printed. A section of bold text in both word processors might look like the following:

DOS:
[b]This is a heading[/b]

Macintosh:
This is a heading

The strive for WYSIWYG can be seen in a number of popular programs. In Adobe Photoshop most dialogs, for instance the Hue/Saturation one, will show what impact your changes will have on your image before you click ok. In the latest version the font list will show what a font looks like, rather than just its name, making it much easier to find the font you’re looking for.
§2.3.11: Flexibility
Flexibility is a broad term. In general it’s about giving the user choices. If you have something that is controlled by a mouse, try making it controllable with the keyboard too. If you have a system in place to draw rectangles it’s usually reasonably trivial to allow drawing circles and triangles. The value of the program will increase significantly however, due to the added flexibility.

§2.3.12: Responsiveness
Computers have a lot of processing power. This means that most operations on a computer are so fast that they appear to be near-instantaneous. The consequence of this is that people have little patience when they are working with computers. They are used to not having to wait for the computer. If certain tasks in your program take more than a few seconds without any visual indication that the program is busy, users will be quick to assume that your program has crashed.

“If the user clicks on a button and your program doesn’t respond in some way within half a second, the user starts getting nervous. One study showed that when the computer failed to visibly respond to a button click, it took just 8.5 seconds for half the participants to assume the machine was hung and press the restart switch.”

§2.3.13: Invisible technology
Most people don’t know exactly what all the parts under their hood are doing when they’re driving a car. They don’t need to. They know a car requires gas and will drive to a gas station when the tank is empty, but they don’t know and shouldn’t need to know exactly how the gas gets transported to the engine and in how many cylinders it gets processed there.

Computers should work the same. Arthur C. Clarke’s Third Law states that “Any sufficiently advanced technology is indistinguishable from magic.”

§2.3.14: Robustness
If a program crashes regularly, users are discouraged to explore. Because they are likely to see an error message whenever they try to do something new, they will only stick to the basic functionality. Obviously this is undesirable. The key then is to make sure the program is robust enough to handle problems. If a problem is unrecoverable, at the very least give them the option to save their work.

“To err is human; forgive by design.”

25
§2.3.15: Protection

One example of this is found in a rather old operating system that had a "daytime" command; users (back in those days, mostly programmers) could type in "daytime" and get the time of day. However, many users in this system also had access to an electronic-mail application. This system used immediate commands — that is, a single character without a "Return" or "Enter" keystroke following it. This supports the guideline to optimize user actions, but this also conflicted with the guideline to prevent user errors.

Sometimes the users forgot whether they were in the operating system or in the mail system. If they typed "daytime" while in the mail system, can you guess what happened? The "d" deleted mail messages, and the "a" made it all messages. Now a good interaction designer would never violate a guideline and allow the user to delete all of anything without confirming it, but the next immediate command the mail system got was a "y" — yes. The "t" meant type out the messages, "i" meant in inverted order, and so on — the rest hardly matters; there were no mail messages to operate on. The final immediate command, "e", meant to exit the email system, making certain that recovery would be impossible.

Occasionally it may be necessary to violate the principle of consistency to prevent potentially destructive actions. A common example is dialog boxes. On dialog boxes it is good practice to always have ok/yes/apply on one side and no/cancel on the other. However, in dialogs like "Are you sure you want to delete all your files?" it might be better to reverse the two so that if the user doesn't read too carefully he will pick the least dangerous option.

§2.3.16: Ease of learning and ease of use

These two apply to two different user groups: the first to new users and the second to people who use the program regularly. They are both very important, but often hard to attain simultaneously. A friendly wizard to, for example, open a new document, will help novices get started, but regular users just want to press Ctrl+N rather than have to click five times to get the same result. Another example is the modifier keys in Photoshop. Holding one or more of the Control, Shift and Alt keys generally changes the function of the tool you are using. This raises productivity, but unless you learn about them in a book or training video you probably wouldn't even know you could use them.
§2.4: Usability testing

§2.4.1: Definition

The general definition for usability\textsuperscript{65} includes five main points:

- Learnability: How easy is it for users to accomplish basic tasks the first time they encounter the design?
- Efficiency: Once users have learned the design, how quickly can they perform tasks?
- Memorability: When users return to the design after a period of not using it, how easily can they re-establish proficiency?
- Errors: How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
- Satisfaction: How pleasant is it to use the design?

None of these points can be deduced without actually letting people test the program. No amount of theories, models, etc. will tell you whether or not people will like your program. It is therefore necessary to conduct usability tests. These have to be done properly though. An incorrectly executed test will at best provide no information at all and in the worst case it will provide wrong information.

This chapter covers some general guidelines for correct usability tests.

§2.4.2: The more, the merrier?

In economics there is the law of diminishing returns. It states that "if one input used in the manufacture of a product is increased while all other inputs remain fixed, a point will eventually be reached at which the input yields progressively smaller increases in output."\textsuperscript{66}

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure15.png}
\caption{Amount of testers vs. percentage of problems found\textsuperscript{67}}
\end{figure}

The same holds true for usability testing. As seen in Figure 15, five to six testers will find 80-90% of your problems\textsuperscript{68}. Hiring an additional ten testers to find a few more problems is inefficient. For this reason it is better to test three iterations of a product with five users each than to test one iteration with 15. Not only will the reduced amount of people prevent the same problems being found numerous times, spreading out the testing will also allow new problems, which may be caused by fixing the old ones, to be found. Another interesting point that can be seen in the graph is that zero testers (i.e., not testing) will not uncover any problems.

One thing to note is that the goal in this study is to find the best use interface, and not specifically to find problems. However, one of the ways of finding the best solution is by eliminating the bad ones. If
five people try a design and four of them aren't able to use it, the design is probably not the best one possible. Naturally, whether or not the design that isn't eliminated is the best one depends on the choice of the tested designs. If all the designs are suboptimal a perfect solution will not be found.

This problem is inherent to all design. Designers only have limited imagination and time, so it is never certain whether the final design is the best one possible. There are no rules or equations to come up with a perfect design, only rules about things that do not work, which are usually based on experience. As the saying goes, “Good judgment comes from experience. Experience comes from bad judgment”.

Fortunately, the goal isn’t to come up with the perfect interface. The goal is to come up with an interface that is easy enough to be used by the majority of people. The method of testing with a limited number of people can therefore be used to eliminate alternatives that work poorly. By having a number of these tests, the interface can iteratively approach the point where most people are able to work with it. By filling the solution space with solutions as they are used in existing software (which presumably have gone through similar tests) and a few new ideas, it should be possible to converge on a good solution.

§2.4.3: The Hawthorne effect
A well known effect in science is the Hawthorne effect, i.e. that the behaviour of people changes when they know they are being studied.

This is of course true in usability tests. The participant is aware that he or she is being watched and will therefore read instructions more carefully. They might have performance anxiety. In general, they will not act the same way as they would normally. It is therefore rather pointless to test things like the total time it takes a user to complete a task. Instead the goal should be to find things that cause problems for the users, like tabbing through text fields not being supported, features not being visible enough so most users don’t know they are there.

§2.4.4: Usability tests don’t measure usability
The term ‘usability test’ is in actuality incorrect, because usability tests don’t measure usability. What they measure is learnability. Usability tests last less than an hour, so it is nearly impossible to determine how a part of the program will be perceived when used day in, day out, for several hours a day. Rather, they test how easy the program is to learn in a short time span. As mentioned in §2.3.16: Ease of learning and ease of use, usability and learnability often conflict. Making a rigorous step-by-step process will improve learnability but hamper usability. This is not necessarily a problem. For applications that will usually be used by new users, e.g. the software on a museum information display, learnability is more important than usability. For professional applications like AutoCAD or Photoshop the opposite is true.

Because the program that is being developed in this project will most likely only be used once or twice when people are buying a house, learnability is more important for this application than usability. This makes usability tests a good way of testing how easy the program is to learn.
§2.5: Program analysis

§2.5.1: Introduction
Before trying to create a user interface, it is always a good idea to analyze other products that are more or less in the same field. This can show good and bad ideas and gives an idea of what kind of interfaces people are used to. It also prevents re-inventing the wheel.

For this application three types of applications will be researched: 2D design programs, 3D design programs and games. The first two are studied to see how other developers allow users to create drawings and 3D models. The latter two are analyzed to see how navigation through a 3D space is commonly handled.

§2.5.2: Software spectrum
Most computer programs fall somewhere in the spectrum that ranges from powerful, but difficult to use to easy, but with limited functionality. The former are generally used by professionals, the latter by average users. Increased difficulty can usually be attributed to an increased amount of commands (see also §2.3.7: Simplicity), which translates in a greater number of toolbars, menus, keyboard shortcuts, etc. that increase the learning curve.

This program should lean towards the simple end of the spectrum, as the target users will not spend significant amounts of time with it.
§2.5.3: 2D design programs

If we look at four popular 2D drawing programs we see a number of similarities. Common features are a viewport in the middle that displays the image being worked on, a tool palette on the left hand side and a menu bar at the top. Both Photoshop (Figure 16) and CorelDraw (Figure 18) feature an extra bar at the top with tool properties and a number of panels on the right with object properties.

Figure 16: Adobe Photoshop

Figure 17: Microsoft Paint

Figure 18: CorelDraw

Figure 19: Autodesk AutoCAD

AutoCAD (Figure 19) is one of the few, if not the only, program still to feature a command-line interface in a time when graphical user interfaces have almost completely replaced the command-line interface. This feature is clearly aimed at professionals, who only need to type one or two letters to access a command instead of having to take several seconds to choose it with a mouse.

Microsoft Paint (Figure 17) on the other hand is a very simple drawing program. There are no extra panels, just the bare essentials required to make a drawing.
§2.5.4: 3D design programs

The main difference between 2D and 3D design programs is of course the extra dimension: this has two consequences for the user interface: extra means of navigation must be provided (it is no longer enough to just pan and zoom, you need to be able to rotate and move the camera as well) and you need be able to view the scene from different viewpoints.

When looking at the screenshots, several similarities with 2D design programs can be seen. The largest interface element is still the viewport. One difference with 2D programs is that in most 3D programs the viewport can be split up into multiple views. One of the more common arrangements is to split it up into a top-down view, a front view, a left or right view and a perspective view. This difference is of course caused by the fact that 3D designs can often not be seen from a single viewpoint. Nearby objects hide other objects, they may be below one another, etc. Some split the viewport up by default, like 3ds Max (Figure 20) and XSI (Figure 27), others have a single viewport by default, like Maya (Figure 21), Blender (Figure 22), SketchUp (Figure 23) and ArchiCAD (Figure 24).

![Figure 20: Autodesk 3ds Max](image)

![Figure 21: Autodesk Maya](image)

![Figure 22: Blender Foundation Blender](image)

![Figure 23: Google SketchUp](image)
The concept of a toolbox is largely intact, though the position tends to vary a lot more than in 2D programs, where the left hand side of the screen is by far the most common one. In 3D programs all four sides of the screen are used. Also interesting to note is that a number of programs opt not to use icons to indicate the tools, but words or larger 3D representations of the objects (Figure 26). This is due to the fact that some shapes and operations are very hard to clearly represent with a small icon.

The navigational controls are usually not very visible, with buttons no larger than any other toolbar or toolbox button. The only exception is Bryce (Figure 25), where the navigational controls take a prominent place in the interface.

Overall the interface is reasonably similar to those of 2D design programs though.
§2.5.5: Games

While 2D and 3D design programs match the functionality reasonably well, the average user will not be familiar with them. 3D programs especially are only used by a very small percentage of the population. For this reason I also look at games, which deal with (more simplistic) design and navigation through a 3D space as well. They have a far higher market penetration than the other two classes of programs, which will only increase over time as people who grew up on video games become old enough to buy a house.

Naturally, there is a vast amount of different game genres. In this analysis I will cover only the ones that most closely resemble parts of the program being developed.

Figure 28: SimCity

Figure 29: The Sims

Figure 30: Black & White

Figure 31: Far Cry
The first game analyzed is the SimCity series (Figure 28). The goal is the same as the program being developed: to allow inexperienced users to make an architectural design, albeit on the urban level rather than on the house level. Again the most important interface elements are the viewport, the toolbox and a property pane. Construction happens on a grid and is done by choosing a tool (e.g., a police station) and clicking where you want to place it. Navigation consists of stepped zooming and rotating in four steps, for which there are interface buttons, and scrolling, which happens by moving the mouse against the edge of the screen or holding down the right mouse button and moving the mouse.

An even better match is the game The Sims (Figure 29), where people can construct their own house. It is one of the best-selling games in history. Like SimCity, which is made by the same company, the game is grid-based and objects are again placed by selecting the object and clicking where it should be placed. The toolbox is at the bottom of the screen this time. Navigation is done with the mouse. Moving the mouse while holding down the right mouse button pans, and moving while holding the middle mouse button rotates the view.

The next game, Black and White (Figure 30), has a rather uncommon interface: apart from a few messages, you interact with the world via a floating hand. To pick up and throw a rock, you hold down the left mouse button over the rock, which picks it up, quickly move the mouse forward to give the rock momentum and let go of the mouse button to release the rock. To move, you hold down the left mouse button over the ground and move the mouse.

Finally, there are two game genres that are relevant. The first is the First-Person Shooter (FPS) genre. The example given is Far Cry (Figure 31), but virtually every game in this genre functions in the same way. The aspect this class of games has in common with the program being developed is first-person navigation through 3D spaces. The way FPS games solve this problem has changed little since the game Quake 2 came out in 1997\textsuperscript{85}. Looking around is done by moving the mouse. Moving the mouse left makes the camera turn left, etc. Walking around is done with the keyboard, usually with the arrow keys or the W, A, S and D keys.

The other genre is that of Real-Time Strategy (RTS) games, here illustrated by Command and Conquer: Generals (Figure 32), where objects are placed from an overhead view. The interface is similar to that of SimCity: objects are placed by selecting the desired object and placing it in the correct location. Clicking on an object will select it and show a pane with the properties of the object. Navigation is often the same as that of The Sims: panning is done by moving the mouse to the edge of the screen or holding down the right mouse button and moving the mouse and rotating is done by moving the mouse while holding down the middle mouse button.
§2.6: Conclusions

The single most important conclusion that can be drawn from the literature research is that the only way to get a user-friendly interface is to base every decision on the user. You must figure out what they want, what they need and what the solution should look like in order to solve their problems in the best possible way.

This idea is shared by Google59, currently the most popular search engine, who include the same sentiment as the first point of their company philosophy60:

"Focus on the user and all else will follow.

From its inception, Google has focused on providing the best user experience possible. While many companies claim to put their customers first, few are able to resist the temptation to make small sacrifices to increase shareholder value. Google has steadfastly refused to make any change that does not offer a benefit to the users who come to the site:

- The interface is clear and simple.
- Pages load instantly.
- Placement in search results is never sold to anyone.
- Advertising on the site must offer relevant content and not be a distraction.

By always placing the interests of the user first, Google has built the most loyal audience on the web. And that growth has come not through TV ad campaigns, but through word of mouth from one satisfied user to another.

As long as you keep focusing on the user all other principles, like consistency, familiarity, etc. will follow automatically.

From the analysis of existing programs a few essential elements can be distilled, as they are so commonplace that ignoring them would be foolish. These are the viewport (the biggest part of the screen which shows the design), the toolbox and a property panel to modify objects.

Since the program to be developed here is aimed at inexperienced users, it should also fall on the low end of the spectrum, i.e. an interface that is as simple as possible. This will restrict the amount of things that can be done with the program, but this is no problem as the users are unlikely to use advanced functionality even if it were included.
Chapter 3: Requirements

§3.1: Introduction

Now that the problem area has been researched a set of requirements that the system should meet can be written. After that, use case diagrams can be created.

§3.2: System requirements

§3.2.1: Non-functional requirements

One of the main goals of the study is to come up with a program that is easy to use for non-expert users. Therefore the system should not require any computer knowledge beyond basic GUI actions like clicking and dragging. Also, no architectural knowledge should be required. If a design is impossible the program should report this. Since users are not expected to have a lot of architectural knowledge, the program should make suggestions to give people ideas or inspiration. These suggestions can come from the designs other people have made or even be generated by the program.

The system should be as intuitive as possible. The user should not be overwhelmed with information on how to use the program, as this will likely turn them off of using the program. A short tutorial might be necessary, but this should not take longer than one or two minutes.

The program is intended for people that wish to buy a house and who have at least some computer experience (e.g., being able to use a mouse). The target demographics will therefore be people of around 25 to 60 years of age. People should be able to build a simple house in about 15-20 minutes. Naturally it will probably take longer to create a house that conforms exactly to their wishes, but a rough version should be able to be produced quickly. While designing the house, the user must be able to see the price of their design while they are building it.

The system is not intended to increase the amount of people that build a house for themselves. It is mainly intended as a replacement for the multiple-choice programs or brochures that are currently being used in housing projects. People who go to an architect themselves don't particularly need this program, as they can have their every wish executed anyway. In those cases the program might be used to reduce the amount of time the architect has to spend discussing with the client, as clients can get a better idea of what they want by playing around with the program.

The intention isn't to contain every single building object on the market. This is infeasible both from a logistical standpoint (every single building product manufacturer would have to agree to have their products included in the program), a technical standpoint (import filters would have to be written for all the different data formats the manufacturers use) and from the user's standpoint. When someone is confronted with a thousand different doors, they are likely to simply pick one that looks reasonable, as it is far too much work to go through the entire list. In practice, the architect would probably make a selection from the building components so customers are presented with a more manageable list.

The system is not a complete replacement for an architect. Architects will still be necessary, but their role will change. Instead of having to design every minor detail of the house, they will make a higher-level design, defining things like the widths of row houses, the materials that can be used in the façade, and making a selection from the list of building products. They will also create a few example designs as
inspiration for the clients. Details like the amount of bedrooms, location of the kitchen, etc. will be left to the clients. Afterwards, the architect will have to check the designs, as it is unlikely the program will check every rule there is. Also, the architect can make suggestions that improve the aesthetics, which are nearly impossible to check for a program.

§3.2.2: Functional requirements

It should be possible to add both products (existing objects that can be bought from a manufacturer) and generic objects (objects that aren't available in reality but that are used as a placeholder until a product is chosen).

Because the price of a design must be visible, the program must have a way to retrieve the current prices of objects, including calculating factors such as labour, transportation, etc.

The products must be defined in an object specification language. Whether this will be a new language or an existing one like IFC or OWL will have to be analysed.

It should be possible to specify rules and constraints at the global, project, house and object level, as well as inter-object relationships. The language for these rules and constraints must be simple enough for the people working with the program, like architects and manufacturers, to use.

§3.3: Use cases

§3.3.1: Prototype use cases

For the prototype being developed in the graduation phase, three main use cases, which represent the most important functions of the program, were identified:

- Navigating around and through a house
  This use case tests whether people are able to navigate through the design, both to change their viewpoint when an objects is in the way and to see the house from a first-person perspective, which allows them to judge what the house would look like in real life.

- Moving a wall and placing a window in it
  This tests the ability to position objects, select a product from the list and place that product in the design. These are the two most frequently performed actions when designing.

- Changing the properties of a generic stair and replacing it with a product
  Tests the method of changing the properties of objects and the concept of having generic objects.

§3.3.2: System use cases

From these three general use cases, the system use cases were derived. The following requirements are all targeted at the level of the entire system, not just at the graduation part of the project. Parts that will not be tested in the graduation phase are coloured grey.

Unless otherwise specified, the default actor is the user and the default prerequisite is that there is an active design open.
§3.3.3: Navigation

In order to visualize the house, the user has to be able to navigate around it. This means they have to be able to move the camera and to switch from an overhead perspective (which is more suitable for designing) to a first-person perspective (which shows people how the house will look in real life). Also, it must be possible to reset the camera to its starting point in case the user becomes disoriented.

The use case diagram is shown in the figure below.

![Use case diagram](image)

**Figure 33: Navigation use case diagram**

<table>
<thead>
<tr>
<th>Use case: Move camera</th>
<th>Possible problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Move forwards, backwards, left or right and look up, down, left or right.</td>
<td>Movement is blocked by something, e.g. a wall. The movement will be stopped.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case: Switch camera</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Click a button to switch between the 1st and 3rd-person camera.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Use case: Reset camera</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Click a button to reset the two cameras to their original positions.</td>
<td></td>
</tr>
</tbody>
</table>
§3.3.4: Products

In order to be able to design a house, it must of course be possible to place objects like walls, windows, etc. People have to be able to select the product they want to place and the location where they want to place it. In addition, the program will support placing generic components, so the user can indicate, for example, that they want to place a one meter wide stair without having to choose a specific product just yet.

![Use case diagram](image)

**Figure 34: Products use case diagram**

### Use case: Find product
1. Select from a list of products or enter search criteria to search for a product

**Possible problems:**
- The product is not found

### Use case: Place product
1. The chosen product is placed in the desired location using the mouse

**Prerequisites:**
- A product has been selected

**Possible problems:**
- The location of the object causes a problem, for instance if the new objects intersects another one. In this case a warning is given

### Use case: Place generic object
1. The chosen object is placed in the desired location using the mouse

**Prerequisites:**
- A generic object type has been selected

**Possible problems:**
- The location of the object causes a problem, for instance if the new objects intersects another one. In this case a warning is given
- The object cannot be placed in the given location, e.g. a window that isn’t placed on a wall. The object will not be placed
Use case: Replace generic object with product

1: The chosen product is placed on the generic object. The generic object is replaced by the product.

Possible problems:
- The location of the object causes a problem, for instance if the new objects intersects another one. In this case a warning is given
- The object cannot be placed in the given location, e.g. a window that isn't placed on a wall. The object will not be placed
- The object can be placed at the chosen location, but no generic object exists to replace. The object will be placed normally.

§3.3.5: Object transformations

The properties of objects will be changed frequently. They are moved to explore different layout possibilities, the material is changed to experiment with different colour schemes, etc.

![Object transformation use case diagram](image)

Figure 35: Object transformation use case diagram

Use case: Select object

1: Click on the object

Use case: Modify object properties

1: The desired property is changed in the property panel that appears when an object is selected.

Prerequisites:
- An object is selected

Possible problems:
- The new property value is outside the valid range (e.g. a negative height). The value is automatically corrected to lie inside the valid range
The selected object is part of a template and has no changeable properties

Use case: Transform object
1: The object is moved or rotated
Possible problems:
- The selected object is part of a template and cannot be transformed

Use case: Remove object
1: Remove an object
Prerequisites:
- An object is selected
Possible problems:
- The selected object is part of a template and cannot be deleted

Use case: Copy object
1: The object is copied
2: The object is pasted
Prerequisites:
- An object is selected

§3.3.6: Identifying and correcting problems
People will occasionally make errors. For this reason most design programs (word processors, graphics applications, etc.) include undo and redo options. Also, some design decisions they make might be suboptimal or even in conflict with building codes or the law. The user has to be made aware of this.

Figure 36: Problems use case diagram

Use case: Undo
1: The user selects Undo from the menu or presses Ctrl+Z. The last action is undone
Possible problems:
- There is no action to undo. Nothing will happen
### Use case: Redo

1: The user selects Redo from the menu or presses Ctrl+Y. The last undone action is redone

Possible problems:
- There is no action to redo. Nothing will happen

### Use case: Get feedback from AI

1: The user makes a change to the design that causes a problem
2: The program indicates that there is a problem

### §3.3.7: Loading and saving

Finally, it has to be possible to open and save designs so people don't have to create the entire design at once but can continue where they left off. Also, the architect has to be able to create templates with objects that cannot be changed, e.g. the width of the house. These templates are opened by the clients.

![Figure 37: Loading and saving use case diagram](image)

### Use case: Load template

1: Choose the project for which you want to design a house
2: Choose the house you wish to design. The appropriate template will be loaded automatically

Prerequisites:
- No scene has to be active to use this function

Possible problems:
- No template exists for the chosen location. An empty scene without any restrictions will be created.
### Use case: Save template

**Actor:** Architect  

1. Choose Save Template from the menu. The current design will become the template for the chosen house  

**Possible problems:**  
- The disk is full or write-protected or there is another file system-related bug. A warning will be given  

### Use case: Load design

1. Choose the Load design option  
2. Select the design you wish to open. The design is loaded  

**Possible problems:**  
- The file is corrupt or of the wrong file type. An error will be displayed  

### Use case: Save design

1. Choose the Save design option. The design is saved  

**Possible problems:**  
- The disk is full or write-protected or there is another file system-related bug. A warning will be given
§4.1: Introduction

As discussed in §2.4: Usability testing, it is important to do usability tests on a program (or, for that matter, anything that needs to be used by customers).

Peter Bickford describes the result of the first user test after he had made a textbox that showed a list of possible options below it while the user was typing (a sort of AutoComplete):

When she got down to the first disambiguating text field on the form, she tabbed into it and the scrolling list popped down. I hadn’t warned her what would happen, and she paused for a moment, saying something like, “Hmm…this is new…” Within a few seconds she had figured out that she could type into the field, and that the scrolling list would follow her selection. “Neat!” she remarked.

I felt vindicated.

Then she hit the return key to “accept” her selection.

Having always known that real users wouldn’t think of using anything but the Tab key to move from field to field, I was utterly unprepared for this, and my program promptly crashed. Having done her duty, the very real Suzanne cheerfully went on her way, and I began coding a new version. My brilliant prototype had survived for less than 30 seconds after being exposed to an actual user.

The second prototype lasted almost a minute before the user, another Administrative Assistant, pressed the Enter key to accept her selection. So it went by all day long. At least six rounds of coding, testing, and recoding went by as I discovered the importance of supporting the arrow keys to move up and down, double-clicking in the list to choose a selection, and so on.

This anecdote illustrates that users frequently do things the developer didn’t expect. During the tests I experienced this myself as well. For instance, I was surprised by the importance of the floor plan and the popularity of dragging with the mouse.

This is why it’s important to do usability tests. As mentioned in §2.3.1: User compatibility, the programmer is not the user (save for the rare instances where the product being developed is a new programming language or development environment), and therefore can never anticipate in what way users will try to use the program. For example, I had included a small position marker on the floor plan to indicate the user’s current location. A lot of the participants tried dragging this dot to move. I hadn’t even considered this possibility so of course it didn’t work. These are the kind of things that only testing with real users can show you.
§4.2: Testing method

During the project, three experiments will be held, each with one prototype. The results of each experiment will determine the look of the next prototype and the alternatives available in it.

All three prototype tests were conducted in the public library of Eindhoven. Logistically, it would have been easier to conduct the test in the university. However, this would result in participants that probably have quite a bit of experience with both design and CAD software, which would not be representative of the target demographic of the program. Therefore the tests were conducted in a public setting, which resulted in participants that better matched the target demographic.

The library could provide participants that were in the proper age category and the idea was that the people there would not have a lot of experience with computers. During the tests this assumption proved to be correct. The average person had little experience beyond Microsoft Word, Internet Explorer and Outlook.

All tests were recorded with a camera so I could focus on what was going on instead of having to take notes (see Figure 38). Additionally, the computer screen was recorded with a computer program so I could also see how they were using the program (see Figure 39).

Figure 38: One of the participants in the second prototype test

The first test was done with 5 participants. The second one had 6 participants and the third test was done with 3 groups of 6 people, because each person only did one third of the test. This was done because the ca. 30 minute test time proved to be too long for a number of people. Reducing the time to about 10 minutes lowered the amount of refusals. Another advantage was that I could get the opinion of more people.

Naturally, because the amount of people tested is rather low, it is impossible to say with complete certainty that certain alternatives are good or bad. However, even with a small amount of people you can still identify trends. If, for example, four out of five people do not figure out how to use keyboard navigation, you can say with reasonable confidence that it is probably not a good idea.
Chapter 5: First experiment

§5.1: Introduction

Once the requirements and use cases had been created (see Chapter 3: Requirements), it was time to start creating prototypes.

In order to get the best results per effort ratio the first experiment was conducted as a paper prototype. A paper prototype consists only of sheets of paper with drawings of the interface. The advantage of this method is that new interface variations can be added in minutes instead of the hours or days that would be necessary to implement them in a real program. This allows testing more alternatives in less time than with a working prototype.

The disadvantage is of course that people can’t actually interact with the prototype. The method is therefore more suited to eliminating bad alternatives than choosing between good ones. Consequently, paper prototypes should be created in the early stages of the project to get a rough idea of the interface. Afterwards, working prototypes should be created to test how the application or product behaves in practice.

For the paper prototype five aspects were chosen to investigate. They were chosen because they are visible most, if not all, of the time and are therefore very important. The five aspects are the product list, navigation, the viewport layout, object transformation and constraints. The choices for each aspect will be discussed in the following paragraphs.

The test results of this experiment can be found in §5.3: Testing.

§5.2: Description of alternatives

§5.2.1: Product list

For the list of products two main alternatives were created. Both are first subdivided by object type and then by manufacturer, as otherwise the length of the list would make it very hard to find the product you’re looking for.
The first alternative (Figure 40) is a treeview component. This was chosen because Windows Explorer uses the same method to display the file system structure. Since Windows is such a ubiquitous operating system it can be expected that a certain percentage of the users of the program use Explorer, and consequently that they are familiar with the concept of showing and hiding subcategories.

The second alternative (Figure 41) is an expanding list that uses more or less the same principle as the treeview, but that uses a different graphical representation. It allows showing more information than the single-line items in the treeview.

Finally, both alternatives are created with and without an explicit “Add this product to the scene” button in order to see which method of adding an object to the scene was preferred.

Other alternatives that were considered are a circle diagram (see Figure 42), which was rejected because it doesn’t scale well if there are large amounts of options, and a menu-based approach which is difficult to convey on paper. This alternative was put off until the second round.

Figure 42: Circle diagram concept
§5.2.2: Navigation

To test the navigation controls three aspects were tested: whether or not navigational UI elements are necessary at all and if so, their location and their appearance.

The first alternative (Figure 43) featured no UI elements for navigation, the idea being that people would use the mouse to navigate.

In the second (Figure 44) and third alternatives the navigational controls were placed inside the viewport, in order to establish that the controls relate to the viewport. Two different designs were created: the first consists of four straight arrows: left, right, forward and backward. The second design replaces the left and right arrows with curved arrows which more accurately depict the fact that the buttons cause the camera to rotate around the house. The zoom buttons were replaced with the commonly used magnifying glass icons, which are used, among others, in Microsoft Word, which is probably one of the most often-used computer programs.

The fourth and fifth (Figure 45) alternatives feature the same button designs, but here the buttons are located in a separate navigation panel (similar to programs like 3ds Max), which allows placing explanatory text near the buttons and reduces the amount of mouse movement required to switch between the buttons.
5.2.3: Viewport layout

Figure 46: One main viewport, small floor plan

Figure 47: Two viewports

Figure 48: Three viewports

The viewport is the biggest interface element in any design application and the one users spend most of their time looking at.

A lot of 3D design applications like 3ds Max and Maya allow you to split the viewport into multiple views. In 3D Studio Max the view is split into a top, front, left and perspective viewport by default.

To see what people preferred, three alternatives were created. In the first (Figure 46), the 3D viewport is the dominant one, similar to SketchUp and games with a minimap, with a small floor plan to indicate the user’s position in the house.

In the second alternative (Figure 47) the floor plan and the 3D view become equally important.

Finally, in the third alternative (Figure 48) the front viewport is displayed as well.
§5.2.4: Object transformation

![Figure 49: Movement and rotation arrows](image1)

![Figure 50: Separate transformation panel](image2)

The two common ways of moving objects in design programs are dragging the object (the preferred method in most 2D and a number of 3D design programs), or using a widget. This is reflected in the alternatives for this part of the program.

The first alternative has no UI elements and expects users to use the mouse to move it.

The second alternative adds an arrow to rotate the object, as rotating is harder to translate into mouse movements than moving.

The third alternative (Figure 49) adds movement arrows as well, resulting in something that looks like the widget found in programs like Maya.

The fourth alternative (Figure 50) places the controls in a separate panel.

§5.2.5: Constraints

![Figure 51: Constraints](image3)

Moving the light gray wall in this prototype (Figure 51) to the left would cause it to intersect the dark gray one. The question is whether this has to be prevented, and if so, in what way. The left and right arrows in this alternative have a different shape than the regular arrows. The idea was that when moving the light gray wall, the other one would move with it, keeping the corner constraint intact.
§5.3: Testing

§5.3.1: Method

The test was conducted in the public library of Eindhoven. Five participants who appeared to be between 40 and 60 years of age were chosen at random from the people visiting the library.

Because in this early stage information gathering is more important than strict alternative selection, the test was conducted as an interview. Since it has a more open structure than a questionnaire it is easier to gather information this way.

The test consisted of five parts. In the first four, they were shown sheets of paper with mock-ups of the program’s interface, one at a time. In each case, they were asked one or two questions about how they would go about accomplishing a task, e.g. how they would navigate around the house. After all sheets in a group had been shown they were all placed on the table, and the participant was asked to rate the alternatives according to preference. The fifth part contained only one sheet of paper. As in the other tests a few questions were asked, but no preference had to be given.

§5.3.2: Design

Unless otherwise noted, the dependant variable is user preference.

Part 1: Product panel
Independent variables:
- Product list organization structure
- Presence of an ‘Add product’ button

Part 2: Navigation
Independent variables:
- Presence of UI elements
- Icons used for navigation buttons
- Control location

Part 3: Viewports
Independent variables:
- The amount of viewports

Part 4: Object transformation
Independent variables:
- Control location
- Amount of controls

Part 5: Constraints
Independent variables:
- Different arrow look

Dependent variables:
- Expected difference in behaviour
§5.3.3: Part 1 - Product panel

First question: How would you get to see the contents of the category “Daken” (roofs)?
Most test subjects said the clicking anywhere on the category item should open it.

Second question: How would you add a product to the design?
Of the participants who were shown the two variants with the “Toevoegen” (add) button (Figure 52, Figure 53) first, almost nobody noticed the button. The subjects who were shown the versions without the button (Figure 54, Figure 55) first did notice it, but most likely because they were looking for a difference from the previous prototype. Even when the button was noticed, it was not always clear. One person expected to have to drag from the add button to the scene. It can be concluded that this is not a good choice.

When asked how to add the product without the button, opinions varied. Some people said clicking the product once adds it to the scene. One person wanted to drag it from the product list to the 3D window. Some people expected to have to draw it in the viewport. Obviously there needs to be a way to indicate this.

A suggestion from one of the participants was to add a description of how to add the object or a button to do so to the list when an item is selected. This is an option worth exploring.
Third question: Which prototype do you prefer? Here a distinction can be seen between the TU/e employees and the general public. The employees preferred the tree structure, as they are familiar with it, whereas the participants in the library preferred the listview with icons. It is not entirely clear whether this is due to the treeview in the prototypes having an additional level (the listview doesn’t have a separate category for the manufacturer), because the listview indents items in a category just like the treeview. The listview will be used unless it proves to be problematic in practice.

One participant proved that care must be taken with the icons for the categories: because the icon for the roofs category had a pitched roof he assumed it didn’t contain flat roofs.

§5.3.4: Part 2 - Navigation

Figure 56: No UI elements

Figure 57: Straight arrows in the viewport

Figure 58: Curved arrows and magnifying glasses

Figure 59: Separate panel, straight arrows
First question: How would you rotate around the house in each of the five prototypes?
The order of the prototypes was varied. The version without any controls (Figure 56) was always shown first. The others alternated between showing the two prototypes with a separate navigation panel (Figure 59, Figure 60) and those with the controls in the viewport (Figure 57, Figure 58) first.

When there were no navigational controls, some subjects assumed it was not possible. Some would try dragging with the mouse. Some suggested making circular movements with the mouse. One person would just use the mouse wheel. Obviously some direction is needed.

The prototypes with a separate navigation panel were generally clear, but not everybody was able to determine the meaning of the icons in the viewport. Especially the version with the arrows caused some different responses, ranging from “to see the back side of the house I would click the up arrow” to “all the arrows point outward, so I can’t rotate around the house”.

Second question: Which prototype do you prefer?
The responses for this question varied a lot. Each of the five subjects in the library picked a different option. However, some remarks can be made. The person that said he preferred the version without controls chose that option provided he knew how to navigate the scene with the mouse. This would require reading documentation, which most people don’t do.

The relatively poor performance of the buttons in the viewport may be due to the fact that it lacks the text explaining the functions of the buttons that the separate panel has. This might be solved by adding tooltips to the buttons in the viewports.

The fact that no preference was concluded on the matter of straight arrows vs. icons indicates the icons can be improved. More icon designs will have to be tested in the working prototype.

A suggestion was made to have controls just below the viewport, to better indicate that the controls belong to the viewport, while not cluttering the viewport with the controls. This solution will be tested in the working prototypes.
§5.3.5: Part 3 - Viewports

First question: What do you think the extra drawing does?
Everybody was able to tell it was a floor plan of the house (first two alternatives) and a front view of the house (third alternative, see Figure 63).

Second question: To which of these viewports do you think you can add products?
This question was asked for all three
Contrary to my personal expectations, in the first alternative (Figure 61) most people assumed they could only add products to the floor plan view, even if it was significantly smaller than the 3D viewport. Some people said you could add products to both. Nobody came up with the idea that you can only add objects to the 3D viewport (my personal idea when making the prototype with the floor plan in a separate panel).

Third question: Which prototype do you prefer?
The second option (a viewport split between the 3D and floor plan view, see Figure 62) was chosen unanimously by the participants in the library. This option will be used for further development.

Two suggestions were made that will be tested: Rotating the floor plan along with the 3D view to maintain a better sense of direction and adding the ability to “maximize” either of the viewports. Both suggestions will be tested in the working prototype.
§5.3.6: Part 4 - Object transformation

First question: How would you move and rotate this table (given no UI controls)?
In this alternative (Figure 64) most subjects would try to drag the table to move it. One person said it was not possible (and it is not entirely sure whether people would be able to figure out that the table can be moved at all without a direct question indicating it could).

Opinions varied more when asked how to rotate it. Some said it was impossible, some expected to click it and get some sort of rotation control, one person would use the right mouse button, and several people would make circular gestures with the mouse. Obviously more direction is needed.

Second question: How would you rotate this table (given a curved arrow)?
Here (Figure 65) most test subjects would click or drag the arrow.

Third question: How would you move this table (given four arrows)?
In this situation (Figure 66) all test subjects would click or drag the arrows to move the table. The one person for who the curved arrow in the previous prototype was unclear said it became clear in this prototype.

Fourth question: How would you move and rotate this table (given a separate control panel)?
All participants would simply click the buttons here (Figure 67).
Fifth question: When you click “Beweeg naar voren” (move forward), which direction does the table go in?
The answer to this question varied from north-west to north to north-east to south. Obviously this method is not clear enough.

Sixth question: Which prototype do you prefer?
The prototype with movement and rotation arrows around the object was preferred by virtually everyone.

In the working prototype the following alternatives will be tested: four mono-directional arrows, two bi-directional arrows above the object, and no arrows, since in practice it will be immediately clear that dragging an object moves it.

§5.3.7: Part 5 - Constraints

![Constraints scene](image)

First question: Do you notice a difference in the arrows?
Everybody noticed that the left and right arrow had a block on them (see Figure 68).

Second question: What do you think the difference is?
The original intent of the arrows (stretching the wall) was chosen by one or two participants. Other interpretations included boundaries, between which the wall could be moved, an indication that the wall could not be moved in that direction, moving the whole assembly instead of just the wall and moving the wall flush against another object.

A different method of showing that the wall can be resized will have to be developed.

Third question: When moving the light grey wall to the left, does the other wall move with it?
The majority of the test subjects did not expect the other wall to move. Two people expected there to be a way to link or unlink the two walls.
§5.3.8: Conclusions

As a result of this test the following choices were made:

For the product panel a list view instead of a treeview will be used. This seems to be the clearer choice. The button to add objects to the scene was often overlooked, so it will be removed.

The viewport will be split in two equal parts, one for the floor plan and one for the 3D view, as people considered the two to be equally important. This was in contrast with my expectation, as I expected that the 3D view would be the most important one.

Navigation will probably happen with arrows around the object, although the no-arrow option will still be included to see how it performs in practice.

No conclusion was reached for the navigation and the current method of constraint indication is unclear. These will have to be tested further.

As a result of this test, the following aspects were added to the list of things to research for the working prototypes:

- The exact layout of the listview
- Different navigational icons
- Navigation controls just below the viewport vs. controls in the viewport
- See if tooltips make a difference in understanding the icons
- Rotating the floor plan along with the 3D view
- Adding an option to “maximize” a viewport
- Four unidirectional movement arrows vs. two bidirectional ones vs. no arrows
- A way of resizing an object
- Check if an option to link/unlink object works
Chapter 6: Second experiment

§6.1: Introduction

After the first experiment most of the aspects of the program that could be tested without interaction had been exhausted. Therefore a working prototype had to be created.

With this prototype, four aspects of the program were tested: the icons of the navigation buttons (since the first experiment had not produced a clear winner), two additional navigation methods that couldn’t be tested without interaction, whether or not the floor plan rotates along with the perspective view (an option that was suggested by one of the participants of the first experiment) and an alternative method of product selection, that again couldn’t be tested without interaction.

The test results of this experiment can be found in §6.4: Testing.

§6.2: Programming language and 3D engine selection

When a programming language was chosen to develop the program in, the following criteria were important:

- Mature support for creating a Graphical User Interface (GUI). This rules out languages like C and C++, where you will have to either use the Windows 32 API directly or the MFC classes, both of which are rather primitive options, and languages that rely on toolkits like GTK or QT, which haven’t completely matured yet.
- Availability of a good quality, free, 3D engine. The goal of the project is not to develop a 3D engine so an existing one will be used to reduce development time. Those rules out most functional languages, like Haskell and Lisp, as 3D engines are almost always developed for imperative languages like C++.

This left the following options:

- Java. I have used Java in my previous project. In it, I created a 3D engine myself, but for this project a pre-existing engine like jME (jMonkeyEngine) could be used. While it is a viable option, I was interested in learning a different language.
- Visual Basic 6 / Delphi. Both languages are reasonably similar. Both allow easy creation of GUIs and both support the 3D engine I ended up using. However, development of both languages has more or less stopped and more powerful languages exist.
- Visual Basic.NET / C#.NET. Creating a GUI is easy in both, and they support all the necessary features, as well as a number of 3D engines. I chose C# as I prefer its syntax over that of Visual Basic. The downside of these two is that they only run on windows.
- Python. Comparable in power to C#. Would also have been an option, but C# appealed to me more.

Out of the 3D engines available for C#, the best one appears to be TrueVision 3D. It is free for non-commercial use and supports all the necessary functionality. The engine is reasonably mature, in active development and is used in commercial applications. There is also an active user group.

The engine hasn’t caused any major problems during the development of the prototype.
§6.3: Description of alternatives

Using the results of the first prototype (which are discussed in), a working program (see Figure 69) was built for the second test. With this, it was possible to test whether people were actually able to use the program, rather than asking "how do you think this feature would work?"

![Figure 69: Second prototype](image)

§6.3.1: Navigation button icons

The first aspect concerned the look of the navigation buttons. Four different variations of the arrows were created, as well as three alternatives for the button that switches between the third- and first-person views (located in the bottom left corner).
§6.3.2: Navigational methods

Figure 70: Mouse navigation

Figure 71: Keyboard navigation

The next test aspect was the navigation method. Four alternatives were created, two of which used navigational buttons. The other two used the mouse (Figure 70), which is used in programs like Rhinoceros, and the keyboard (Figure 71), which is inspired by games.

§6.3.3: Floor plan rotation

The next aspect came from a suggestion from one of the participants in the first test. He suggested rotating the floor plan along with the 3D view for easier orientation. In one alternative this was turned on, in the other the floor plan didn’t rotate.

§6.3.4: Product selection

Figure 72: Product list

Figure 73: Product menu

The final test aspect was the method of selecting products. Because there was a working program at this point, the menu alternative (Figure 73) could be compared with the list alternative (Figure 72).
§6.4: Testing

§6.4.1: Method
The test was again conducted in the public library of Eindhoven. As in the first test, five participants were chosen at random from the people visiting the library, this time between 30 and 60 years of age.

An interview structure was again chosen in order to obtain more information than it would be possible to get with a questionnaire.

The test consisted of three parts, with the first two split into three subparts each. The test was conducted on a laptop, with the participants using a working prototype of the program. In each test they were asked to perform a task, e.g. navigating around the house, in between two and four alternatives. Having done that, they were asked which of the alternatives had their preference.

§6.4.2: Design
In each case the dependent variable is user preference.

Part 1: Navigation outside
Independent variables:
- Navigation button icon
- Navigation method (buttons, keyboard or mouse)
- Navigation button locations

Part 2: Navigation inside
Independent variables:
- Camera switch icon
- Navigation button icon
- Navigation method (buttons, keyboard or mouse)

Part 3: Product selection
Independent variables:
- Product selection method
§6.4.3: Navigation outside
The first part of the test dealt with navigating around the house.

§6.4.3.1: Part 1 - Navigation icons

The first four alternatives were about the four icons in the bottom left corner.

Alternative A (Figure 74) and D (Figure 77) were equally preferred with two votes each. One person chose alternative C (Figure 76), saying she preferred it to alternative B (Figure 75).

In general it can be concluded that curved arrows are preferred to straight arrows. This was expected, as comments in the first prototype test had indicated as much already.
The second set of alternatives dealt with the way people navigate around the house. The first two alternatives were based on buttons, in alternative K (Figure 80) the mouse is used and in alternative L (Figure 81) you navigate with the keyboard.

The first two alternatives posed no problems for anyone. The other two however failed almost completely. There was one person that figured out alternative K in a few seconds (the idea was to drag the mouse in a viewport while holding the left or right mouse button and using the mouse wheel to zoom), but everybody else tried to click in vain on the explanatory graphic, apparently not recognizing that the icons meant to move the mouse. Alternative L, where the idea was to use the keyboard, suffered a similar fate. Everyone just clicked on the graphic, not seeing the connection with the keyboard. The
person that figured out the mouse navigation eventually figured it out in the ‘Navigation Inside’ part of the test.

We can obviously eliminate alternatives K and L as possible solutions. Almost nobody understood them and allowing people to click on the graphic would just result in inferior versions of the first two alternatives. Of the remaining two alternative J (Figure 79) was generally preferred over alternative H (Figure 78).

§6.4.3.3: Part 3 – Floor plan

This part dealt with whether or not the floor plan should rotate along with the 3D view, based on a suggestion I received during the first prototype test.

Alternative Q (Figure 83), where the floor plan rotated, was unanimously preferred over alternative P (Figure 82).
§6.4.4: Navigation inside

The second part of the test was walking around inside the house. As mentioned in the general conclusions, this part was a spectacular failure. Only two people actually succeeded in moving from the initial position. Different alternatives will have to be developed for the final round of prototyping.

§6.4.4.1: Part 1 - Switching icons

Figure 84: Alternative E (outside)  Figure 85: Alternative E (inside)  Figure 86: Alternative F (outside)

Figure 87: Alternative F (inside)  Figure 88: Alternative G (outside)  Figure 89: Alternative G (inside)

Two people preferred alternative G (Figure 88, Figure 89), one person wanted a combination of the house from alternative F (Figure 86, Figure 87) and the footsteps of alternative G and one person chose option E (Figure 84, Figure 85).

The suggestion to combine F and G seems like a good one. The bird might not be clear to everyone, but the footsteps clearly won out over the other options. Combining the house and the footsteps should give the best results.
§6.4.4.2: Part 2 - Navigation icons

Since there was only one person that managed to walk through the house who answered the questions, the significance of this part can be doubted. However, given the fact that that person, together with two others, preferred alternative D (Figure 93), which was also one of the two preferred options from the first part of the test, it can be concluded that alternative D is the favourite. Another reason for this conclusion is that alternative D was the only one where the person that managed to walk through the house figured out he could use the buttons to walk instead of the mouse wheel. Nobody associated the zoom icons used in alternatives A (Figure 90), B (Figure 91) and C (Figure 92), which posed no problems outside, with walking.

It also appeared that some people thought that the arrows for looking up and down were for moving around. In the third prototype I will therefore try grouping the movement and left and right arrows and relocating and de-emphasizing the buttons for looking up and down. This should also reduce the time some people spent looking at the floor or ceiling in the 3D view.
Again, because only one person actually walked through the house the results of this part aren’t entirely conclusive. The person that managed to complete the task preferred alternative L (Figure 97), as using the keyboard was faster than using the buttons. Given that nobody else was able to figure out that L implied using the keyboard however this option is not suitable. Alternative K (Figure 96) was similarly unclear. Among the others alternative J (Figure 95) was generally preferred over alternative H (Figure 94).

For the final round I plan on creating an alternative combining alternative J and L: in the bottom left corner four buttons will be laid out in the inverted T-shape of the keyboard arrow buttons, with up and down having the move forward/backward icons of alternative D. Left and right will be the rotate left/right buttons, again from alternative D. This way people can click the buttons and those that realize that they can use the keyboard as well can do so. The look up/down buttons will be placed in the top left corner and the button to switch between inside and outside will be put in one of the corners on the right hand side. This way all the controls are on the edge as much as possible, leaving more workspace for the 3D model than the current alternatives.

Another important point that came out of the test was that most people, when instructed to walk through the house, tried dragging the red circle that indicates the person’s position in the floor plan with the mouse. This didn’t work in the prototype of course, as dragging was tied to rotation. In the
final prototype I will have to make sure people can navigate this way, as it seems to be a common method for people.

One last remark the person that walked through the house made is that he would’ve liked it to be easier to move through the house at 90-degree angles. Once you rotate it is nearly impossible to return to an orthogonal direction. This will be taken into consideration for the final prototype, possible via snapping.

§6.4.5: Product selection

Alternative M (Figure 98) is a list with a tree structure. N (Figure 99) is a standard menu like you can find in almost any application, only with bigger icons, that appeared when clicking on a button.

Although nobody had any problems with either option, alternative M was preferred by all but one person. According to some people the list was a more familiar way of organizing objects to them than the menu structure.
§6.4.6: Conclusions

For the navigation method, alternative J (the 3x3 group of buttons) will be chosen. Mouse and keyboard navigation were a complete failure and alternative J was preferred over alternative H (two bars with buttons).

The icons will consist of the curved arrows and magnifying glasses. These were generally the clearest. This leaves alternatives A and D, of which D was rated slightly better. These icons will be used from now on. A combination of the bird from alternative G and the house from alternative F seems like a good combination, but further testing will be required.

Having the floor plan rotate along with the 3D view (alternative Q) was unanimously preferred.

Finally, displaying the products in a list was preferred over using a menu.

In this test it became apparent that the length of the test (ca. 20 to 30 minutes) was a problem for a lot of people. As a result of this, the third test was split up into three parts to cut the duration of the test down to about 10 minutes.

While doing this test I realized that the application was not only useful for older people, as 25-40 year olds need houses just as much as 40-60 year olds. It was therefore decided to widen the target demographic to 25-60 year olds.
Chapter 7: Third experiment

§7.1: Introduction

With the results of the second experiment, the prototype could be developed further.

In this experiment, the three use cases, as defined in §3.3.1: Prototype use cases, were tested. For this, the following aspects were added to the prototype: the layout of the navigational buttons, as there had been some confusion in the second experiment between the ‘loop up’ and ‘move forward’ buttons, various ways of moving objects, ways of indicating constraints and warnings, the ability to (re)place objects and finally a way of changing the properties (like, width, height, etc.) of objects.

The test results of this experiment can be found in §7.3: Testing.

§7.2: Description of alternatives

§7.2.1: Navigation buttons

Because the second prototype test revealed some issues with the layout of the navigation buttons, an alternative was created (Figure 100) that would make a greater distinction between the move and rotate buttons.

Figure 100: The new navigation button layout

Additionally, three more icon sets were generated for the button that switches cameras.
§7.2.2: Working with objects

The major addition in this version was the ability to place and move objects like stairs and walls.

![Figure 101: One of the alternatives to move a wall](image)

In order to determine the best interface for moving objects, three alternatives were created: two variations of arrows surrounding the object (see Figure 101) and one alternative with no interface elements.
§7.2.3: Constraints and warnings

Not every move action is necessarily a valid one. For instance, a user might move a window outside of a wall, creating a floating window. Naturally this is undesirable behaviour, so something must be done to prevent this. There are two possible times to do so: either stop the object from moving when it would be placed in an illegal position (constraints), or allow the movement and alert the user to the problem (error messages). Both alternatives were tested. For the constraints option both an explicit (a lock icon is shown over the object) and an implicit (stop the object, but don’t display anything) version were created.

![Figure 102: Explicit constraint](image)

![Figure 103: Icon-based warning](image)

The warning system had three alternatives: in one a traffic light indicated the validity of the design (Figure 102), with a tooltip listing all the problems, the second featured warning icons over the problematic objects in question (Figure 103), similar to some computer games, and the third one consisted of a list of problems on the right-hand side.

§7.2.4: (Re)placing objects

The method of (re)placing objects was copied from the one that is found in most games (see §2.5.5: Games): you select the object you wish to place and then click on the location where you want to place it.

§7.2.5: Changing properties

In order to change the properties of objects (length, width, etc.) a PropertyGrid control was chosen, as a lot of 3D design programs (e.g. AutoCAD and 3D Studio) use that control or something similar to change object parameters. This component was visible already in the second prototype, but was disabled as objects hadn’t been implemented yet.
§7.3: Testing

§7.3.1: Method

The test was again conducted in the public library of Eindhoven. This time 18 participants between 30 and 60 years of age were chosen at random from the people visiting the library.

Again, an interview structure was chosen in order to obtain more information than it would be possible to get with a questionnaire.

The test consisted of three parts, each comprised of three or four subparts. The three tests had six variations each, in which the order in which the alternatives were presented was switched to eliminate this as a variable. As in the second tests, participants were asked to take place behind a laptop and use a working prototype of the program. Each subtest again consisted of the participant being given a task, such as moving a wall, and them trying to accomplish it in one to four alternatives, after which they were asked to indicate which of the alternatives had their preference.

§7.3.2: Design

In each case the dependant variable is user preference.

Part 1: Navigation
Independent variables:
- Navigation button layout
- Camera switch icons

Part 2: Navigation Moving and placing objects
Independent variables:
- Presence of UI elements
- Amount and location of UI elements
- Presence of a grid
- Presence of constraints
- Indication of constraints

Part 3: Product selection
Independent variables:
- Method of indicating warnings
- Method of indicating generic objects
- Presence of auto-correction of problems
§7.3.3: Use case 1 – Navigation

§7.3.3.1: Part A – Button layout

![Alternative A1](image1.png) ![Alternative A2](image2.png)

None of the six participants had trouble with the assignment. Four people preferred A1 (Figure 104), because the buttons are closer together, resulting in less mouse movement. Two people chose A2 (Figure 105), one of them because according to her it had fewer options, saying she didn’t know the functions of all the buttons. Presumably she did not see the button in the upper right corner in alternative A2, as both alternatives have the same number of buttons.

Alternative A1 is the favourite in this test.

§7.3.3.2: Part B – Icons

![Alternative B1](image3.png) ![Alternative B2](image4.png) ![Alternative B3](image5.png)

Three people had no problems switching between the interior and the exterior view. One person switched successfully, but said he didn’t understand how to get to the front yard. When told he is in the front yard he said he thought he was at the other side of the house. The problem here might be the rather Spartan scene. Perhaps if the scene has been more detailed, for instance with trees or other orientation points it would have been clearer. One person couldn’t find how to switch when alternative A2 was selected. When switching to A1 she saw the button and switched successfully. Again it appears the switch button in the top right corner gets overlooked easily. One person could not figure out how to switch views.

As for the icons, opinions were divided. One person chose B1 (Figure 106); B2 (Figure 107) and B3 (Figure 108) were both preferred by two people. Both people that chose B2 mentioned the term “helicopter view”. One of the people who chose B3 on the other hand thought the plane was unclear.
More tests will have to be done to determine a clear winner. Possible variants include replacing the plane in B2 with a chopper, and combining a house icon with the walking man from B2.

§7.3.3.3: Part C – Walking through the house

All but one person had no problems walking through the house. One of them initially walked by moving the position marker (the red dot) around, but preferred the buttons when told they could be used as well. Contrary to expectation, four out of five people chose alternative A1 (Figure 109) over alternative A2 (Figure 110), again saying that they preferred having all the buttons together. Apparently the order in which the alternatives were presented resulted in the poor performance of this alternative in the previous test.

Alternative A1 was the clear favourite in this test.
§7.3.4: Use case 2 – Moving and placing objects

§7.3.4.1: Part A – Moving a wall

Everybody managed to move the wall. One person had some trouble differentiating the wall and the floor and it took a bit of time for her to succeed, the rest could move it almost immediately. Four people never used the arrows: they just dragged the wall. This was reflected in the choices for favourite alternative: C3 (Figure 113) got four votes, one person chose C1 (Figure 111) and one person had no preference. Alternative C2 (Figure 112) proved to be unpopular.

The choice for alternative D (Figure 114) was clear as well: five out of six people preferred to have it on, some of who stated is easier to move objects to a specific location with snapping on.

The winners are C3 and turning D on.
§7.3.4.2: Part B – Selecting and placing a window

While four people managed to place the window in the wall (Figure 115), the fact that two of them did it more by accident makes the success rate rather low. The way placing objects was implemented was to select the product on the pane on the right and then clicking where you wanted to place it. Selecting the desired window wasn’t a problem for anyone. When trying to place the window two people followed the intended actions of moving the cursor to the wall and clicking. The two people that succeeded by luck tried to drag the window in the wall (which wasn’t implemented), but happened to click after dragging, placing the window. The two people that didn’t succeed both tried double-clicking on the window.

Some more testing will have to be done. One thing that has to be implemented is dragging objects, which proved even more popular in the third test. Also the cursor change (adding a white plus sign to the bottom right of the cursor arrow) might have been too subtle. A variant where a picture of the chosen object is attached to the cursor will have to be tested. This might also make dragging clearer.
Like the wall moving the window poses no problems for anyone. This time alternative C3 (Figure 118) is unanimously preferred over C1 (Figure 116) and C2 (Figure 117).

Obviously C3 will be the chosen way of moving objects from now on.
Only one person saw a need to move a window outside of a wall: one person chose E1 (Figure 119), two people chose E2 (Figure 120) and three people chose E3 (Figure 121). The general response from those who chose E3 was that it was clear enough that the window was being blocked so they didn’t need the lock.

E1 loses, but there is no clear winner between E2 and E3. However, the people that chose E3 didn’t have any strong feelings towards E2; they just didn’t need it. Those that chose E2 thought it was clearer. This combined with the fact that in test 3 a similar method of drawing in icon over an object when there is a problem is preferred leads to selecting E2 as the winner. It increases clarity of what’s happening for some people and others are merely indifferent to it.
§7.3.5: Use case 3 – Generic objects

§7.3.5.1: Part A – Changing the properties of a stair

Three people succeed in changing the properties of the stair (see Figure 122), one after trying a number of other options. The other three give up. Three people (two people who failed and the person that tried a number of other options first) appear to overlook the property pane. It turns out a lot of people focus on the viewport and don’t really notice the things next to it. This is confirmed in the rest of the test. One person notices it but doesn’t realize she can change the values. Three people try to use the arrows to change the size of the stair.

Clearly the property grid is not a good solution. Two alternatives come to mind: One is a property grid-like control, but with icons to draw attention away from the viewport and ‘spinners’ (text boxes with two small arrows to increase or decrease values) instead of plain textboxes to both indicate that the value can be changed and to get around the problem inherent to the property grid that you have to press enter or click somewhere else to confirm the change in value. The other is using the arrows to resize objects, since they are no longer needed for movement, as proven in test 2.

These alternatives will have to be tested further.

Figure 122: Stair properties scene
The traffic light in F1 (Figure 123) was unanimously overlooked. Four people chose F3, as it gives an indication in the viewport. The other two preferred a combination of F2 (Figure 124) and F3 (Figure 125): they wanted both notifications in the viewport and a list of all the problems.

The combination of F2 and F3 is the winner. It has the traffic sign that everyone likes and people will either like, ignore or never even notice the list of problems. It combines the best of both worlds: you can see which object causes the problem but you don’t have to look around the entire house to find all the problems. You could also implement that clicking on a problem in the list will take you to the chosen problem for more efficient problem solving.
§7.3.5.3: Part C – Generic objects vs. products

Figure 126: Alternative G1

Figure 127: Alternative G2

Figure 128: Alternative G3

Again the warning sign is the most popular choice. Four people prefer G3 (Figure 128), one person prefers a combination of G1 (Figure 126) and G3 and one person prefers G2 (Figure 127).

G3 is the clear winner. It is consistent with the other warnings and actually lets you know there is a problem, unlike G1 and G2.
Replacing the stair turned out to be quite hard. The actions required are the same as in part B of test 2, where people are instructed to place a window. The result here was even worse than before: nobody succeeded in replacing the stair. Remarkably, dragging was even more popular than before: four people tried to drag the stair to the new location. This took me by surprise, as I had always considered dragging a somewhat advanced action, given the difficulties I've seen some people have performing it. Apparently it is a relatively common action for placing and moving objects.

Once the stair was placed F3 (Figure 131) again proved to be the favourite option. Like before, four people choose F3 and two people choose a combination of F2 (Figure 130) and F3. Three people explicitly stated they didn’t like F4 (Figure 132). They want the program to alert them something is wrong, not attempt to fix the problem itself. The traffic light in F1 (Figure 129) was often missed.
As for (re)placing objects, subsequent prototypes will have to implement the drag action, which would significantly increase the success ratio in this test. With regards to displaying problems a combination of F2 and F3 is the winner again, for the same reasons as in part B.

§7.3.6: Conclusion

As for the alternatives, there were a number of clear winners. A1 (all navigation buttons close together), C3 (no arrows around an object), D (snapping and grid on) and G3 (produce warnings for generic objects) were all obvious favourites. The choice for E2 (preventing a window from being placed outside a wall and displaying a lock when doing so) and F2 and F3 (showing warning signs on the objects and having a list of all problems) were also not very hard.

The choices that require additional testing are alternative B (the icons for switching between the interior and exterior view), the method of changing the properties of an object and the method of (re)placing objects. This will be done in the PhD phase of the project.

One general conclusion is one that I also came across in my literature study: people don’t read. Despite the instructions comprising no more than one sheet of A4 I frequently had to correct people who skipped parts of the test. The conclusion is that the less text you need to use, the better.

Another conclusion is that user tests need to be as short as possible. Going from 30 minutes to 10 minutes certainly lowered the refusal rate, but even 10 minutes proved to be a problem for some people. Ideally tests where you approach random subjects should probably take five minutes or less. However, at some point in my project I will have to test if people can actually design a house with the program. This will take considerably more than five minutes. For these tests I will have to consider using a different testing method, for instance inviting people to the university so they have more time. The best solution for this will have to be considered.
Chapter 8: Conclusions and future research

§8.1: Introduction
The goal in this graduation project was to develop a set of interface requirements for a system that allows people to design their own house. These requirements will be used in the PhD phase of the project to create the interface for the prototype that will be developed.

First, a literature study was done in order to study related research and to find general guidelines for user interfaces. Additionally, popular commercial design programs were analysed to find common paradigms. After the literature study a set of requirements was created for the program.

In order to test the interface, three prototypes were created, each of which was tested in an experiment. Each prototype tested a number of aspects of the program and offered a set of alternatives for every aspect. The participants of the test were then asked to use the various alternatives and to indicate their presence. If an alternative proved to work well it was used in the next prototype. If no clear conclusion was reached, the alternative was explored further in the next experiment.

In this chapter the conclusions of the research are presented.

§8.2: Incremental testing
First of all, my research showed that it is essential to conduct user tests for any piece of software being developed. Without testing, it is virtually impossible to create a program that is easy to use.

As for the testing method: the philosophy of conducting regular, small tests instead of a few big tests proved to be a good one. By testing features before a lot of time is spent on them you can save yourself a lot of work if people don’t like or understand it. This method is called “Fail early”. It states that if something is going to fail anyway, it should happen as soon as possible. For example, in the paper prototype phase the amount of viewports was tested. Two viewports, one for the floor plan and one for the 3D view, was the clear winner. The time spent on the other alternatives was no more than a few minutes, as they only had to be drawn, not implemented. This probably saved an hour or so of having to implement them all and make sure switching between them works correctly.

Having said that, paper prototypes (which offer the best ratio of effort spent vs. the amount of eliminated options) are not perfect. You are trying to test an interface, which means the customers have to interact with the program. You can ask hundreds of “How do you think this would work?” questions and still end up with a bad interface, simply because theory is no replacement for practice. For most projects paper prototypes are probably only useful for one to maybe three iterations. After that functioning prototypes will have to be built.

In functioning prototypes it takes a lot more effort to create alternatives. Even here it is possible to increase the effect per effort ratio though. When adding a new feature, it doesn’t have to work in every situation, just the one you are testing. For instance, the walls in my prototype work as long as they are parallel to the x-axis. Rotating the wall would cause it to fail, which is the reason the rotation arrow was left out of the prototype. For the test you are doing it makes no difference, and it saves a lot of time coming up with a completely correct solution. The goal should be to only implement complex things completely if you are sure it is going to stay in the program.
§8.3: Finalized interface elements

Product view
Grid
Viewports
Property pane
Object transformation
Warnings
Constraints
List of problems
Navigation controls
§8.3.1: Viewports
The amount of viewports was already established during the paper prototype phase. In practice the two-viewport option seemed to work well as well. The second test showed that having the floor plan rotate along with the 3D view was the preferred option.

§8.3.2: Grid
The grid was preferred by most people and because this will eliminate a lot of off-by-a-few-millimetre problems, as well as make things like measurements cleaner to display, it will be used from now on.

§8.3.3: Object transformation
Most people chose the alternative without arrows and they did not use them when they were displayed. Therefore this variant will be chosen. The arrows might be used for scaling instead of translation. This will be researched.

§8.3.4: Constraints
Having constraints was preferred over having no constraints. People were either positive or neutral about showing a lock icon when movement was constrained, so this will be the chosen method.

§8.3.5: Navigation
In the final test almost nobody had any problems with navigation, so the current button-based method, as well as the button icons, will be maintained. The only thing that will be studied further is the look of the button that switched between first- and third person mode, as that can still be improved a bit.

§8.3.6: Product view
The basic functionality of the product panel is finalized, as most people seem to have little trouble with it. On the visual front a test will be done to see if triangular arrows or plus and minus buttons improve clarity.

§8.3.7: Property pane
The current implementation of the property pane is not conspicuous enough. Not everyone figured out immediately how to use it. More research will have to be done to improve ease of use and conspicuousness.

§8.3.8: Warnings
Problems will be displayed both via an icon over the problematic object, as well as in an overview list of all the problems in the scene. The list of problems will have to be made more conspicuous though, as it was overlooked several times in the last test.
§8.4: Non-finalized interface elements

The chosen method of placing a selected building object proved to be unsuccessful. However, since nearly everyone chose the same method (dragging the object from the list to the viewport), it is clear what the alternative should be. This will have to be implemented.

The property grid that is used for changing things like the width of a stair is not conspicuous enough. A number of people missed it. One solution is to use colour or graphic elements to make it stand out more. Alternatively, the arrows around objects could be used, as they are no longer necessary for movement. This will be tested.

§8.5: Comparison with existing software

In this paragraph the results from the prototype tests are compared with the way the software packages discussed in §2.5: Program analysis work.

§8.5.1: Viewports

The most common viewport configurations in commercial programs are one 3D viewport (2D design programs, games and some 3D design programs) or four viewports (top, left/right, front and perspective; mostly found in more professional 3D design programs). It is interesting then that the preferred option here was two viewports. I suspect this results from the fact that houses have always been presented to customers via an artist’s impression (3D) and a set of floor plans (2D).

In the final test most people used the 3D viewport as the working viewport, and referred to the 2D viewport to see their location in the house while walking around. A few people used the 2D viewport for manipulating objects as well.

§8.5.2: Grid

Grids used to be commonplace in computer games, because they could be easily represented on computer screens. Since ever more powerful graphics cards have made displaying fully 3D worlds a possibility however, the grid is slowly being phased out. Of the analysed games only the Sim series of games (SimCity, The Sims) still has a grid. The next game of the Sim series’ creator Will Wright, Spore, does away with the grid as well. The main games to still have grids are on less powerful platforms like portable consoles, and some strategy games.

The other class of program that displays grids by default are the professional 3D packages like 3D Studio, Maya and SoftImage. Snap is off by default though, and they may serve more as a plane of reference than as an actual grid.

That the grid was preferred in this application may be explained by the fact that the exact position of objects is more important here. If a building in a game is a few millimetres too far left it will not have any consequences. In a house it may mean that there is a hole in the wall. A grid makes it easier to position objects exactly.

§8.5.3: Object transformation

The chosen option of having no arrows around an object can be seen in 2D design programs as well. When selecting an object in Photoshop or CorelDraw you only see handles for scaling the object. Moving the object is done by dragging, the same as the option that was selected in this program. The
only difference is that in commercial programs the cursor changes to the move cursor, which may be a good idea to implement.

§8.5.4: Constraints
Most 2D and 3D design programs don’t really apply constraints to a design. The two most similar situations probably occur in games. When trying to walk through a wall in a First-Person Shooter game movement is simply blocked. When trying to place an object over another one in the Sim series, the object being moved turns red. This is an option that hasn’t been tested yet. In a future prototype this will be compared to the currently selected option of blocking movement and showing an icon.

§8.5.5: Navigation
In 2D design programs navigation is mostly mouse-driven, usually by using a pan tool. 3D design programs add orbit and zoom tools to this, but work in mostly the same way. In games the mouse is the main navigation tool as well, supplemented by the keyboard in FPS games.

It is surprising therefore that the mouse alternative was such a failure. The most likely explanation is that the participants had no experience with computer games and that the only design programs they work with are the Microsoft office products, where navigation is mainly done with scroll bars, which are more or less comparable to navigation buttons.

§8.5.6: Product view
In games product lists are more commonly displayed as large icons, with textual information showing up in a popup balloon. These lists however are usually limited to about 20 items, and the items are different enough to be easily recognised. In this program however it is not unthinkable that for example a list of doors will have three different models in ten colours each. Seeing the difference between oak and maple wood in these icons is more difficult, and having to hold the mouse over each one to see what the material is would quickly become annoying.

Therefore I think that the current option of showing an icon and a short description, which expands to the detailed description when the item is selected, is a good one.

§8.5.7: Property pane
The property pane as it has been implemented so far in the prototypes can mainly be found in professional 2D and 3D design programs. In games that have something similar (mostly Real-Time Strategy games) the property pane usually consists of large icon buttons. Something similar may be needed to improve the visibility of the control, as it was often overlooked. In future prototypes alternatives will be tested.

§8.5.8: Warnings
The icon-based warnings in the viewport are probably the most common way of indicating warnings in video games. Examples include Unreal Tournament, Command & Conquer and The Sims. As mentioned design programs have little or no constraints, so warnings are virtually non-existent as well.
§8.6: Goal achievement

The goal of this project was to define a set of interface requirements for the final system. This goal has been achieved: the interface has proven to be reasonably successful.

There are still a few areas that need to be studied further, like using the arrows around objects for changing their size, making the whole object red when it violates a rule, improving the properties pane, testing the dragging method that proved to be popular for placing objects and some other minor tweaks.

Overall the system seems to be usable for non-experts.

§8.7: Points of improvement

An obvious thing that might be improved is the amount of alternatives. In this study only a limited amount of alternatives were implemented due to time constraints. It is quite possible that there are better ideas that haven't been tested. The alternatives in this study were largely based on ideas found in commercial software, as those ideas have undergone a similar process and are likely better than a lot of other alternatives. They are based on the same process though: they are one of the options their designers came up with, which may or may not contain the best possible solution. It is doubtful whether you can claim that a certain design is the best at all, as it is impossible to test every possible solution.

On a similar note, the study has focused on a traditional computer with a keyboard and a mouse. This was chosen due to logistical issues and because the participants are at least somewhat familiar with them, removing some of the anxiety they might have when faced with a completely unfamiliar system like a CAVE. However, alternative systems or input devices like a CAVE or a 3D joystick might theoretically be better solutions. This would have to be tested.

Another improvement might be to test with more participants. While the current method of testing only with a limited amount of people proved successful at eliminating the poorer alternatives, selecting between the better ones might have benefited from a larger amount of people. This would allow statistical analysis rather than majority rule. Again, this would require more resources.

Finally, as noted in §7.3.6: Conclusion, people often skipped parts of the test. Some people had to be corrected several times. This might have been prevented by reducing the amount of text even further or by increasing the font size or the spacing between the questions.
§8.8: Planning for PhD phase

The PhD phase will consist of implementing the logic behind the program, consisting of:

- The object library
  Because the design is to be made with commercial products, the program naturally needs to have a database of these products, with information like the manufacturer, cost and the 3D representation of the object. Due to the vast amount of existing building components it is infeasible to create this list myself. There are two possible solutions to this:
  - One option is to contact these companies and see if it is possible to incorporate their library in my program. The most likely candidate for this is the firm De Twee Snoeken, who have an extensive product library as part of their ZoekSnoek application. Additionally, the company collaborates with the university on a regular basis.
  - An alternative to this is to use the semantic web approach. The two main existing systems for this are IFC and OWL. These are probably overkill for the intended purpose here though, so if this method is the chosen the most likely approach is specifying an own format that is compatible with either of these two formats, but that only uses a subset of the possibilities they offer so the format is easier to handle.

- Rules and constraints
  One of the main differences between the approach of this program and existing 3D modelling software will be the possibility of applying rules and constraints to the design, like the minimum amount of windows in a room. These are necessary to ensure that the design doesn’t have any major problems before the design is sent to the architect for a more detailed inspection. It must be easy to edit, create and remove rules so that the program can be updated to changing laws or site-specific constraints without a lot of work.

- User dialogue
  As mentioned before, the people using the program are not professional designers. They will therefore probably miss a lot of possibilities. The program must therefore be able to suggest options the user hadn’t considered yet. The suggestions can come from the designs of other users or the program might even be able to generate some alternatives itself.

Additionally, further user interface tests will be necessary to improve the aspects that were not yet clear after the third prototype test, as well as testing existing features, like object warnings, when the logic behind them works correctly, so they can be tested in a real house design instead of a small scene of two or three objects.

§8.9: Planned platform

The current application was developed in C# 2.0 using the TrueVision 3D engine. This has worked quite well. One option therefore is to keep using C# (though switching to the new 3.0 version) and the TrueVision. The alternative is Vizard, a 3D engine for Python. The basic style of the two languages is largely the same and the two 3D engines seem roughly equally powerful. At the start of the PhD phase I will therefore try porting the current program to both Python and C# 3.0 to see if either platform offers significant advantages over the other (like GUI programming probably being easier in C# than in Python, or Vizard possessing a physics engine), and if there no major advantages, to see which one appeals to me more.
References

5. Appropriateness and impact of platform-based product development, V. Krishnan and S. Gupta, 2001, Management Science 47 (1), pp. 52-68
10. Developing a customer service culture in the speculative housebuilding industry, D. Craig and R. Roay, 1999, Warwick Manufacturing Group, University of Warwick, Mimeo
13. Platform driven innovation in Construction, J.I.M. Halman, 2004, Faculty of Engineering Technology, University of Twente, Enschede
15. STUDIO ODE for advanced presentations | indexpagina, www.studio-ode.nl
17. TNO – kennis voor zaken, www.tno.nl
23. CBS StatLine, statline.cbs.nl
30. IAI and IFC – state-of-the-art, M.A. Lacasse, D.J. Vanier, 1999, Information technology in construction, volume 4, Vancouver

93
Only
72 Autodesk - Autodesk 3ds Max, www.autodesk.com/3dsmax
73 Autodesk - Autodesk Maya, www.autodesk.com/maya
74 blender.org - Home, www.blender.org
75 SketchUp Home, www.sketchup.com
76 Graphisoft - Welcome to ArchiCAD 11!, www.graphisoft.com/products/archicad/ac10/
77 DAZ Productions - 3D Models, 3D Content, and 3D Software, www.daz3d.com/i.x/software/bryce/-/
80 SimCity™ 4 - SimCity.com, simcity.ea.com/
81 The Sims™ - Official Site - Find your Sims community, thesims.ea.com/
82 Black & White, www.lionhead.com/bw2/
83 FarCry > News > Current, www.farcry-thegame.com/
88 Devmaster.net – 3D Game and Graphics Engines Database, A. Busygin, www.devmaster.net/engines/
89 Truevision3D :: 3D Engine and Game Development SDK, www.truevision3d.com
Further reading

Books


About Face 2.0: the essentials of interaction design, A. Cooper and R. Reimann, 2003, Wiley, Indianapolis


Software Engineering, I. Sommerville, 1996, Addison-Wesley, Amsterdam

User-interface screen design, W.O. Galitz, 1993, QED, Wellesley

Articles and papers


Appendix A1 – First round questionnaire
Prototype interview

Prototype interview parameters
The interviews will be held in a public place, like a library or city hall instead of on the university campus in order to ensure the tested people do not already have a lot of experience with computers.
The goal will be to interview around 10 people of around 30-60 years of age.
The interviews will be held with paper prototypes.
The interviews will be videotaped.
If the person interviewed does not answer a question ‘correctly’ or doesn’t know, no hints will be provided.

Prototype interview schedule

- Ask someone to participate in a 3-5 minute study
- Explain the program to them (it will allow them to design a house themselves)
- Ask them for age, profession, and computer experience
- Explain the goal of the study (research interface) and explain that not being able to give the ‘right’ answer is a fault of the interface, not of them.
- First question: Object panel. The order of the questions will be alternated.
  o 1: Show first and second prototype (treeview and listview with add button)
  o How would you switch to a different object category?
  o If the list becomes too long, how would you shorten it?
  o How would you add an object to the scene?
  o 2: Show third and fourth prototypes (treeview and listview without add button)
  o How would you add an object to the scene?
  o Please sort the prototypes according to preference
- Second question: Navigation. The order of the questions will be alternated between 1-2-3-4 and 1-4-2-3.
  o 1: Show first prototype (no navigational UI elements)
  o How would you make sure you could see the other side of the house?
  o 2: Show second prototype (arrows in viewport)
  o What do you think the buttons do?
  o 3: Show third prototype (icons in viewport)
  o What do you think the buttons do?
  o 4: Show fourth and fifth prototype (arrows and icons in separate panel)
  o What do you think the buttons do?
  o Please sort the prototypes according to preference
- Third question: Viewports. The order of the questions will be alternated between 1-2-3 and 2-3-1.
  o 1: Show first prototype (plan view in separate panel)
  o What do you think this does?
  o Do you think you can add objects to it as well?
  o 2: Show second prototype (plan view in viewport)
  o Do you think you can add objects to the plan view?
  o 3: Show third prototype (plan and front view in viewport)
  o What do you think the new part is?
  o Please sort the prototypes according to preference
• Fourth question: Object manipulation. The order of the questions will be alternated between 1-2-3-4 and 1-4-2-3.
  o 1: Show first prototype (table)
  o How would you move the table?
  o How would you rotate the table?
  o 2: Show second prototype (table with rotation control)
  o How would you rotate the table?
  o 3: Show third prototype (table with movement and rotation control)
  o How would you move the table?
  o 4: Show fourth prototype (controls in separate panel)
  o What direction would the table move if you choose 'Move left'?
  o What if you rotated the view?
  o Please sort the prototypes according to preference
• Fifth question: Constraints
  o Do you notice anything different with the controls?
  o What do you think the change means?
  o If you use the left arrow button, what will happen to the light grey wall?
  o What will happen to the dark grey wall?
• Thank the interviewed person for their time
Inleiding
Geachte heer/mevrouw,
Hartelijk dank voor uw medewerking aan dit onderzoek. Het onderzoek gaat over een programma waarmee mensen zelf hun woning kunnen ontwerpen. In deze test worden een aantal alternatieven getest voor hoe het programma ongeveer gaat werken.

In de test zal van u gevraagd worden een aantal taken te verrichten. Hierbij zal geen hulp geboden worden. De bedoeling is om te kijken of de alternatieven intuitief genoeg zijn. Mocht u niet weten hoe u een bepaalde taak moet voltooien, zeg dit dan gerust en ga verder met de volgende taak. Dit is geen enkel probleem en geeft slechts aan dat het alternatief niet intuitief genoeg is.

Er zal een aantal keer naar uw voorkeur voor een alternatief gevraagd worden. Aangezien de test op video wordt opgenomen kunt u simpelweg uw voorkeur uitspreken.

Mocht u op- of aanmerkingen hebben op de alternatieven, uiteenzet dan gerust.

Gelieve het programmavensneter niet te verplaatsen of groter of kleiner te maken, aangezien alleen dit deel van het scherm wordt opgenomen.

Deel 1: Navigatie buiten
De eerste test gaat over het navigeren buiten het huis

1. Probeer om het huis heen te draaien en in en uit te zoomen.
2. Klik in het menu op Alternatieven en vervolgens op Variant B. Probeer om het huis heen te draaien en in en uit te zoomen.

Welke van deze alternatieven heeft uw voorkeur?

8. Klik in het menu op Alternatieven en vervolgens op Variant L. Probeer om het huis heen te draaien en in en uit te zoomen.

Welke van deze alternatieven heeft uw voorkeur?


Welke van deze alternatieven heeft uw voorkeur?
Deel 2: Navigatie binnen
De tweede test gaat over het door het huis heen lopen.

1. Klik in het menu op Alternatieven en vervolgens op Variant H.  
2. Probeer te schakelen tussen het binnen- en buitenaanzicht van het huis.  
3. Klik in het menu op Alternatieven en vervolgens op Variant F.  
   Probeer te schakelen tussen het binnen- en buitenaanzicht van het huis.  
4. Klik in het menu op Alternatieven en vervolgens op Variant G.  
   Probeer te schakelen tussen het binnen- en buitenaanzicht van het huis.  

Welke van deze alternatieven heeft uw voorkeur?

5. Schakel naar het binnenaanzicht van het huis.  
6. Klik in het menu op Alternatieven en vervolgens op Variant A.  
   Probeer door het huis heen te lopen.  
7. Klik in het menu op Alternatieven en vervolgens op Variant B.  
   Probeer door het huis heen te lopen.  
8. Klik in het menu op Alternatieven en vervolgens op Variant C.  
   Probeer door het huis heen te lopen.  
9. Klik in het menu op Alternatieven en vervolgens op Variant D.  
   Probeer door het huis heen te lopen.  

Welke van deze alternatieven heeft uw voorkeur?

10. Klik in het menu op Alternatieven en vervolgens de gekozen variant uit de  
    vorige vraag. Loop door het huis heen.  
11. Klik in het menu op Alternatieven en vervolgens op Variant J.  
    Probeer door het huis heen te lopen.  
12. Klik in het menu op Alternatieven en vervolgens op Variant K.  
    Probeer door het huis heen te lopen.  
13. Klik in het menu op Alternatieven en vervolgens op Variant L.  
    Probeer door het huis heen te lopen.  

Welke van deze alternatieven heeft uw voorkeur?

Deel 3: Productoverzicht
De derde test gaat over het productoverzicht

1. Klik in het menu op Alternatieven en vervolgens op Variant H.  
2. Probeer een product te selecteren.  
3. Klik in het menu op Alternatieven en vervolgens op Variant N.  
   Probeer een product te selecteren.  

Welke van deze alternatieven heeft uw voorkeur?

De test is voltooid. Hartelijk dank voor uw medewerking.  
Indien u nog op- of aanmerkingen heeft, meldt deze dan gerust.
**Test 1 – Variant 1**

**Inleiding**
Geachte heer/mevrouw,

Hartelijk dank voor uw medewerking aan dit onderzoek. Het doel van het onderzoek is het testen van een programma waarmee u uw eigen huis kunt ontwerpen.

In de test zult u steeds gevraagd worden een bepaalde taak te verrichten in twee tot vier varianten. Indien u een opdracht niet kunt voltooien omdat het niet lukt of omdat u niet weet hoe het moet, ga dan door naar het volgende alternatief. Dit is geen enkel probleem en geeft aan dat het alternatief niet duidelijk genoeg is.

De bedoeling is om na elk onderdeel aan te geven aan welke van de alternatieven u de voorkeur geeft. De test wordt op video opgenomen dus u kunt uw voorkeur gewoon hardop uitspreken. Tevens kunt u op- of aanmerkingen plaatsen en suggesties ter verbetering voorstellen.

**Onderdeel A: knoppenlayout**
U kijkt momenteel tegen een kant van het huis aan. Probeer de andere kant van het huis in beeld te krijgen (d.w.z. probeer om het huis heen te draaien). Probeer in- en uit te zoomen (d.w.z. het huis van dichterbij en verder weg te bekijken).

Klik in de menubalk op Alternatieven en vervolgens op A2.

Probeer opnieuw om het huis heen te draaien.

Welke van deze alternatieven heeft uw voorkeur?

**Onderdeel B: iconen**
Kies voor alternatief A1.


Herhaal dit voor alternatief B3.

**Onderdeel C: door het huis lopen**
Schakel naar het binnenaanzicht van het huis. Probeer door het huis heen te lopen.

Kies voor alternatief A2 en probeer opnieuw door het huis heen te lopen.

Hiermee is de eerste test voltooid. Indien u belangstelling heeft kunt u ook meedoen aan de tweede en/of derde test. Tevens bent u welkom om commentaar of suggesties te geven.

Hartelijk bedankt voor uw medewerking.
Test 2 – Variant 1

Inleiding
Geachte heer/mevrouw,

Hartelijk dank voor uw medewerking aan dit onderzoek. Het doel van het onderzoek is het testen van een programma waarmee u uw eigen huis kunt ontwerpen.

In de test zult u steeds gevraagd worden een bepaalde taak te verrichten in twee tot vier varianten. Indien u een opdracht niet kunt voltooien omdat het niet lukt of omdat u niet weet hoe het moet, ga dan door naar het volgende alternatief. Dit is geen enkel probleem en geeft aan dat het alternatief niet duidelijk genoeg is.

De bedoeling is om na elk onderdeel aan te geven aan welke van de alternatieven u de voorkeur geeft. De test wordt op video opgenomen dus u kunt uw voorkeur gewoon hardop uitspreken. Tevens kunt u op- of aanmerkingen plaatsen en suggesties ter verbetering voorstellen.

Onderdeel A: een muur verplaatsen
Probeer de muur die u ziet te verplaatsen.

Klik in de menubalk op Alternatieven en vervolgens op C2. Probeer de muur opnieuw te verplaatsen.

Kies voor alternatief C3 en probeer de muur opnieuw te verplaatsen.

Welke van de alternatieven heeft uw voorkeur?
Zet alternatief D aan en probeer opnieuw de muur te verplaatsen. Is dit een verbetering?

Onderdeel B: een raam selecteren en plaatsen
Zet alternatief D uit. Probeer een raam te kiezen en dit in de muur te plaatsen.

Onderdeel C: een raam verplaatsen
Kies voor alternatief C1. Probeer het raam te verplaatsen.

Herhaal dit voor alternatieven C2 en C3.

Onderdeel D: beperkingen
Probeer het raam te verplaatsen zodat het deels uit de muur steekt. Plaats het raam terug in de muur.

Kies voor alternatief E2 en probeer opnieuw het raam buiten de muur en weer terug te plaatsen.

Herhaal dit voor alternatief E3.

Hiermee is de tweede test voltooid. Indien u belangstelling heeft kunt u ook meedoen aan de eerste en/of derde test. Tevens bent u welkom om commentaar of suggesties te geven.

Hartelijk bedankt voor uw medewerking.
Test 3 – Variant 1

Inleiding
Geachte heer/mevrouw,

Hartelijk dank voor uw medewerking aan dit onderzoek. Het doel van het onderzoek is het testen van een programma waarmee u uw eigen huis kunt ontwerpen.

In de test zult u steeds gevraagd worden een bepaalde taak te verrichten in twee tot vier varianten. Indien u een opdracht niet kunt voltooien omdat het niet lukt of omdat u niet weet hoe het moet, ga dan door naar het volgende alternatief. Dit is geen enkel probleem en geeft aan dat het alternatief niet duidelijk genoeg is.

De bedoeling is om na elk onderdeel aan te geven aan welke van de alternatieven u de voorkeur geeft. De test wordt op video opgenomen dus u kunt uw voorkeur gewoon hardop uitspreken. Tevens kunt u op- of aanmerkingen plaatsen en suggesties ter verbetering voorstellen.

Onderdeel A: eigenschappen van een trap veranderen
Probeer de eigenschappen van de trap, zoals breedte en hoogte, te veranderen.

Onderdeel B: waarschuwingen
Maak de trap 50 centimeter breed.
Klik in de menubalk op Alternatieven en vervolgens op F2. Kijk naar het verschil.
Kies voor alternatief F3 en let weer op het verschil.
Welke van de alternatieven heeft uw voorkeur?

Onderdeel B: producten vs. algemene objecten
Kies onder Alternatieven voor Scene 4.
U ziet twee trappen. Een van de twee is een product met vaste maten, de andere is een algemene trap waarvan u de maten kunt veranderen zoals in onderdeel A.
Kijk naar het verschil in weergave van de producttrap en de algemene trap.
Kijk ook naar het verschil tussen de twee trappen in de alternatieven G2 en G3.

Onderdeel C: een algemeen object vervangen door een product
Kies voor Scene 5 en alternatief F1 en G1.
Probeer de algemene trap te vervangen door een product en bekijk het gevolg
Bekijk het verschil ook in de alternatieven F2, F3 en F4.

Hiermee is de derde test voltooid. Indien u belangstelling heeft kunt u ook meedoen aan de eerste en/of tweede test. Tevens bent u welkom om commentaar of suggesties te geven.

Hartelijk bedankt voor uw medewerking.
Execution of the program starts in mainForm. In mainForm the 3D engine, the program loop and the input class are instantiated. The Engine class handles everything related to the 3D scene, like creating and deleting objects and rendering the scene. The Input class handles user and mouse input that is related to navigating. The ProgramLoop continually reads user input, updates the scene accordingly and renders the scene. The mainForm class also initializes a Design.
The design holds all the building objects in the scene, as well as the cameras.

BuildingObjects are defined by a 3D mesh. ParametricBuildingObjects are generated dynamically. In the prototype three parametric building objects are implemented.
All the building problems are held in a list.

Finally, there are a few classes that are used throughout the program (especially Vector, as it is used in almost everything related to 3D).