

The translation of ambiguous client requirements into product specifications

By: Masoud Delghandi

Master Thesis

Construction Management and Engineering
Eindhoven University of Technology

February, 2018

Date final presentation: February 27, 2018

Supervisors:

Prof. Dr. Ir. B. (Bauke) de Vries – TU/e

Ir. Ing. A. (Aant) van der Zee – TU/e

Ir. W. (Wiet) Mazairac – TU/e

This page is intentionally left blank

Colophon

General

Report : The translation ambiguous client requirements into product specifications

Date : February 20, 2018

Date presentation : February 27, 2018

Place : Eindhoven

Student

Author : Masoud Delghandi

Student number : 0831557

E-mail : m.delghandi@student.tue.nl

: masouddelghandi@live.nl

University : Eindhoven University of Technology

Master Track : Construction Management and Engineering

Chair : Information Systems in the Built Environment

Supervisors

Chairman

Prof. Dr. Ir. B. (Bauke) de Vries – TU/e

First supervisor

Ir. Ing. A. (Aant) van der Zee – TU/e

Second supervisor

Ir. W. (Wiet) Mazairac – TU/e

Eindhoven University of Technology

Den Doldech 2

Postbus 513

5612AZ Eindhoven

www.tue.nl



This page is intentionally left blank

Preface

This thesis forms the last task that I was required to produce, present, and defend to successfully finish the Master track in Construction Management and Engineering at Eindhoven University of Technology. After finishing my first degree in Building Engineering at the ID-college in Gouda on the Secondary Vocational Educational level, I found that I fell in love with a variety of topics related to the domain of Architecture, Engineering and Construction. This feeling challenged me to attend at The Hague University of Applied Science to achieve a bachelor degree in the Built Environment. The more knowledge I gained, the more I found that crucial knowledge was missing. This induced me to attend to the Master track Construction Management and Engineering (CME) at Eindhoven University of Technology. I found that the Master Track CME could offer me knowledge on the variety of interests that I have within the domain of Architecture, Engineering and Construction. This especially to the topics of building information management, process management, project management, urban planning, legal and governance, and information systems within the built environment. The curriculum of the Master track gave me the opportunity to select the courses in which I wanted to learn or excel in. Of course, during both the Master and this graduation process, a lot of fellow humans helped me out by achieving my results. That's why I would like to take a moment to show my sincere gratitude for these persons.

I felt inspired from the first until the last moment that I got in contact with Prof. Dr. Ir. B. (Bauke) de Vries. Professor de Vries provided me always with the right information and knowledge and always challenged me during the mentoring conversations. His patience, guidance and expertise were of great importance both during my development as a Master student and this final graduation project. Before Prof. Dr. Dipl. Ing. J. (Jakob) Beetz was leaving the Eindhoven University of Technology, I was lucky enough to be guided under his supervision during a half year Research and Development project. During this project, he challenged and revealed me the level of competence that is required to contribute and solve problems for mankind. He inspired and taught me how to get familiar with computer science and programming for Architecture, Construction and Engineering, and what level of effort and dedication it takes to be ahead of the curve in this field. During my Master I attended to some courses where I found Ir. A. (Aant) van der Zee being related to the topics of Architecture, Building, Planning, and automation. A few semesters deep, I experienced that Ir. A. (Aant) van der Zee was more than familiar with programming of all sorts of things. I found that he could guide and teach me how communication with computer experts should be in order to introduce and initiate automation for certain engineering purposes. Ir. A. (Aant) van der Zee taught me the difference in how humans think they are specific, what being specific really is and requires, and how this relates to Construction Management and Engineering. This was what initiated this research initiative at first.

Ir. T. (Thomas) Krijnen, thank you for that morning that we were sitting and talking about the possibilities that I had to transform my ideas in a computer program, this conversation inspired and motivated me a lot. I also want to show my sincere gratitude to Ir. W. (Wiet) Mazairac for being that flexible to attend to my graduation commission given his busy schedule. I felt blessed to have a roomy such as you Kay Wortel. You, as Math and Computer Science student stood next to me during my crave yard shifts to get this program in Java code up and running.

Finally, I want to thank my mother and father for bringing my brother and me over to the Netherlands since 1993. You both took a lot of calculated risks for us. Yours and my brothers' education, loyalty, patience, courage, guts, guidance, and energy is nowhere written in books and took me where and what I am today.

I hope that this thesis report will contribute to the domain of Construction Management and Engineering, to science in general, and that you as a reader will have an amusing and instructive experience while reading this.

Masoud Delghandi

This page is intentionally left blank

Contents

Contents	3
1. Glossary	7
1.1 List of figures	7
1.2 List of tables	10
1.3 List of abbreviations	11
Abstract	13
Summary	15
Samenvatting.....	17
2 Introduction.....	19
2.1 Motivation	20
2.2 Problem definition.....	22
2.3 Research scope.....	23
2.4 Importance	24
2.5 Related work	24
2.6 Primary hypothesis & objective	26
2.7 Research questions	26
2.8 Research design.....	27
2.9 Expected results	29
3 Literature review.....	31
3.1 Motivation	31
3.2 Design process.....	32
3.2.1 Information exchange in the design process	35
3.2.2 Systems engineering	37
3.2.3 Design phases.....	43
3.2.4 Requirements.....	45
3.2.5 Verification.....	50
3.2.6 Conclusions	51
3.3 Knowledge Management	53
3.3.1 From data to knowledge	56
3.3.2 Conclusion.....	58
3.4 Natural language constraints	60

3.4.1	Constraints within engineering	60
3.4.2	Methods of using constraints	62
3.4.3	Constraint entry	62
3.4.4	Conclusion	66
4	In-house practices	67
4.1	Motivation	67
4.2	Interview	67
4.3	Definition of subjects	70
4.4	Interview results	70
4.4.1	Design process	71
4.4.2	Interpretation of requirements	72
4.4.3	Verification.....	79
4.4.4	Requirement classification.....	79
4.4.5	Automation of translation procedure	80
5	Model	83
5.1	Method.....	83
5.1.1	Evolutionary prototyping	83
5.1.2	System requirements	85
5.2	Use Case(s)	86
5.2.1	Use case 1: TRANSLATE	86
5.2.2	Use case 2: Database Manager BOK	88
5.3	Prototyping process	90
5.4	System operational functionality	92
6	Results	95
6.1.1	The Bank of Knowledge application.....	95
6.1.2	The Graphical User Interface	95
6.1.3	Lexical analysis	96
6.1.4	Word enrichment.....	96
6.1.5	Word allocation.....	97
6.1.6	Requirement specification	98
6.1.7	Database and input	99
7	Procedures for system use.....	105
7.1	Procedure for the translation of a mono-disciplinary requirement, Use case 1: TRANSLATE.	105
7.2	Procedure for the translation of a non-functional requirement, Use case 1: TRANSLATE.	110

7.3	Procedure for database management, Use case 2: Database Management	114
8	Conclusion	127
9	Recommendations	135
9.1	Recommendations for implementation	135
9.2	Recommendations for future research & development	136
10	References	139
11	Appendices	145
11.1	Appendix A: Program design	145
11.1.1	building.BufferedReaderPlus	146
11.1.2	building.log.Log.....	148
11.1.3	building.GenerateDesign	148
11.1.4	building.WordDef	149
11.1.5	building.Advisor	150
11.1.6	building.editor.DefFileEditor	156
11.1.7	building.editor.Sort.....	160
11.1.8	building.editor.Def.....	161
11.1.9	building.editor.WordCell	162
11.1.10	buidling.editor.WordCellGroup	163
11.2	Appendix A: building.BufferedReaderPlus	165
11.3	Appendix B: building.log.Log	173
11.4	Appendix C: building.GenerateDesign.....	174
11.5	Appendix D: building.WordDef.....	178
11.6	Appendix E: building.Advisor	179
11.7	Appendix F: building.editor.DefFileEditor	182
11.8	Appendix G: building.editor.Def.....	201
11.9	Appendix H: building.editor.Sort.....	206
11.10	Appendix I: building.editor.WordCell.....	211
11.11	Appendix J: building.editor.WordCellGroup	214

This page is intentionally left blank

1. Glossary

1.1 List of figures

Figure 1: Research model.	28
Figure 2: Project life cycle of a construction project with the project phases based upon BNA et al., 2009; Eadie, Browne, Odeyinka, Mckeown, & McNiff, 2013; Nederlands Normalisatie-instituut, 1993.	33
Figure 3: Iterative character during the design phase (Moonen, 2016)	34
Figure 4: Macleamy Curve (Lu, Fung, Liang & Rowlinson, 2015).	35
Figure 5: Interaction between requirements and design solutions.	36
Figure 6: Systems engineering process (US de partment of defense, 2001).	41
Figure 7: V-Model Systems engineering process extended, based on (Moonen, 2016; BAMinfra, 2008; INCOSE, 2015; Werkgroep Leidraad Systems Engineering, 2007).	42
Figure 8: Interaction in design process, based upon Schaap et al., 2008.	43
Figure 9: Hierarchy of client needs, based upon (Walraven & de Vries, 2009).	45
Figure 10: Interaction between information in requirements and objects (Moonen, 2016).	48
Figure 11: Requirement type classification (Moonen, 2016).	49
Figure 12: Essence of the verification process (Moonen, 2016).	50
Figure 13: Activity diagram 1: The interpretation, translation and verification process.	74
Figure 14: Activity diagram 2: Assessing requirements on SMART principle.	75
Figure 15: Activity diagram 2.1: Dissection of requirement into interfaces.	76
Figure 16: Activity diagram 3: Verification planning.	77
Figure 17: Activity diagram 4: Verification.	78
Figure 18: Evolutionary prototyping process.	84
Figure 19: Use case 1, use case diagram: TRANSLATE, Bank of Knowledge.	86
Figure 20: Use case 1, Activity diagram: TRANSLATE, Bank of Knowledge.	87
Figure 21: Use case 2, use case diagram: Database Manager BOK.	88
Figure 22: Use case 2, Activity diagram: Database Manager BOK.	89
Figure 23: The graphical user interface of the Bank of Knowledge system (Use case 1).	95
Figure 24: Tokenization of word, definition(s), class(es) and specification(s).	99
Figure 25: The graphical user interface of the 'Database Manager BOK' (Use case 2).	100
Figure 26: The variety of definition domains for word(s) enrichment.	101
Figure 27: The variety of predefined classes as a function of the SBS.	101
Figure 28: The structure of a specification.	101
Figure 29: Fundamental structure of an enriched token.	102

Figure 30: Data structure within the database.	102
Figure 31: Formal representation of a token within the database.....	102
Figure 32: Formal representation of a token enriched with HTML web links.	103
Figure 33: Input of a mono-disciplinary requirement within the ‘requirement input field’	105
Figure 34: The enriched representation of the obtained words within the ‘word enrichment’ field... ..	106
Figure 35: The definition of the word ‘smoke detector’, within the ‘word enrichment field’, is defined by means of a Web Based thesaurus (Art & Architecture Thesaurus, 2017). This knowledge is captured within the system its database.	107
Figure 36: The definition of the word ‘equivalent’, within the ‘word enrichment field’, is defined by means of a Web Based thesaurus (Visuwords WordNet, Princeton 2017). This knowledge is captured within the system its database.	107
Figure 37: The allocation of the word ‘Siemens fd0221’ and ‘equivalent’ on a subsystem level within the SBS according to the NL-SfB as presented within the ‘word to object allocation’ field. Here, both ‘Siemens fd0221’ and ‘equivalent’ are allocated to the NL-SfB class no. 60. The specification of the ‘Siemens fd0221’ is expressed by the HTML link named as ‘initial product specification’, and ‘equivalent’ is expressed in specifications as obtained from ‘Vendor A’ – ‘Vendor B’ – and ‘Vendor C’ that deliver product with the same specification or better.	108
Figure 38: The valid product specification that communicate the product performance’s as obtained from the vendor’s website which are provided within the ‘word to object allocation specification’ column as a web link (figure X). ..	109
Figure 39: Input of a non-functional, somehow fuzzy, requirement within the ‘requirement input field’.	110
Figure 40: The enriched representation of the obtained words within the ‘word enrichment’ field.....	111
Figure 41: The allocation of the word ‘warm’ as obtained from the non-functional requirement on a subsystem level. In this example, ‘warm’ has been automatically allocated to ‘40-Finishing’, 50-Services mainly mechanical, and 60-services mainly electrical within the SBS within the ‘NL-SfB classification’ field.....	112
Figure 42: The valid product specification that communicate the product performance’s obtained systems database which are provided within the ‘word to object allocation specification’ (figure X).....	113
Figure 43: The descriptions of the operational functionality that are accommodated within the graphical user interface of the ‘Database Manager BOK’ (Use case 2).....	114
Figure 44: The fundamental ‘linguistic definition’ of the token ‘cold’.....	115
Figure 45: The allocation of the ‘token’ on a system level by means of a ‘NL-SfB class’ and its ‘NL-SfB class definition’	116
Figure 46: The allocation of the ‘token’, with the word ‘cold, on a system level by means of a second ‘NL-SfB class’ and its ‘NL-SfB class definition’ do declare ambiguity regarding the definition and interpretation of the word ‘cold’. ..	117
Figure 47: The ‘specification’ of the token in terms of ‘product performances’	118
Figure 48: The translation of the token ‘cold’ in product specifications by means of the BOK system.....	119
Figure 49: The findings of the ‘google search engine’ after entering the search keys ‘warm room colors’.	120
Figure 50: Fragment of the findings of the google search on the keyword ‘Banana’, very small sample size. Here, we will not find apples or peaches within its findings.....	121

Figure 51: Fragment of the findings of the google search on the keyword ‘Rectangular’, very small sample size. Here, we will not find circles or triangles within the findings. **121**

Figure 52: The findings of the ‘google search engine’ after entering the search keys ‘warm room colors’. **122**

Figure 53: Photo sample as obtained from the google search on ‘warm room colors’. **122**

Figure 54: Extracting a color sample of the obtained picture to capture knowledge on colors that have been defined by individuals on the globe as ‘warm room colors’. **123**

Figure 55: The color sample that defines ‘warm room colors’ in terms of TIH and RGB specifications. These specifications can be converted to any kind of color scales that product manufacturers use. **123**

Figure 56: The final representation of the mono-disciplinary requirement, the specification, is stated within the ‘word to object allocation specification’ screen. **124**

Figure 57: UML class diagram of the system **101**

1.2 List of tables

Table 1: Typ of requirements based upon (Schneider & Berenbach, 2013).	45
Table 2: Description of analyzed projects (Moonen, 2016).	49
Table 3: Data analysis outcome (Moonen, 2016).	49
Table 4: SECI model of knowledge conversion (Nonaka, 1991).	57
Table 5: Use case 1: Use case text; Translation of (fuzzy) requirement.	87
Table 6: Use case 2: Use case text; CRUD (Create – Read – Update – Delete).....	89

1.3 List of abbreviations

3D	three dimensional
AEC	Architecture, Engineering and Construction
BDA	Big Data Analytics
BDE	Big Data Engineering
BIM	Building Information Modelling
BNA	Bond Nederlandse Architecten
CAD	Computer Aided Design
CB-NL	Nederlandse Concepten Bibliotheek voor de bouw
CROW	Centrum voor Regelgeving en Onderzoek in Grond-, Water-, en Wegenbouw en de Verkeerstechniek
CSV	Comma Separated Values
DBFMO	Design Build Finance Maintain Operate
DM	Design Management
D&E	Designers and Engineers
ES	Expert System
FBS	Function Breakdown Structure
GUI	Graphical User Interface
IC	Integrated Contract
ICT	Information and Communication Technology
IFC	Industry Foundation Classes
INCOSE	International Council on Systems Engineering
ISO	International Organization for Standardization
KBS	Knowledge Based System
KDD	Knowledge Discovery in Database
KM	Knowledge Management
LOD	Level of Detail
ML	Machine Learning
NEN	Normalisatie en Normen
NL-SfB	Elementenmethoden
OBS	Object Breakdown Structure
OTL	Object Type Library
SBS	System Breakdown Structure
SE	Systems Engineering
SMART	Specific, Measurable, Attainable, Realizable and Time bounded
SUHA	Simple Uniform Hashing Assumption
UAC-IC 2005	Uniformed Administrative Conditions for Integrated Contracts
UAV-GC 2005	Uniforme Administratieve Voorwaarden voor Geïntegreerde Contracten
UI	User Interface

This page is intentionally left blank

The translation of ambiguous client requirements into product specifications

Masoud Delghandi

Construction Management and Engineering, Eindhoven University of Technology

Keywords: Building Information Management, Requirement Management and Engineering, Design Management, Systems Engineering, Client specific requirements, Physical requirements, Functional requirements, Non-functional requirements, Automation in construction.

Abstract

The rising complexity of demand specifications formulated by unprofessional clients within the AEC-industry induces Designers and Engineers (D&E) to adjust their strategies in regards to Design Management (DM). Inexperienced clients are often not technically skilled. This makes it very hard to capture the right interpretation of the client's intentions for a specific requirement by D&E prior to the formulation of product specifications that satisfy the demand. This research focuses on the translation procedures of quantitative and qualitative client specific requirements into product specifications for conceptual design stages. The objective of this research initiative is to explore the possibilities to introduce automation as a technique to optimize these reoccurring translation procedures in regards to effectiveness and efficiency.

Within this research, a literature study was conducted on the topics of the design process, Systems engineering, Knowledge Management and Natural Language Constraints. The findings from the review of literature were merged with the observations obtained from interviews held with specialists from the field of Systems Engineering. Based on the findings from these methods, a methodology has been developed. This method has been accommodated by means of an evolutionary prototyping process within a software program, named as 'The Bank of Knowledge'. This program, on the one hand, accommodates a method that can translate both quantitative and qualitative client requirements into product specifications by means of automation. On the other hand, this program provides a digital environment in which words that are extracted from client requirements can be stored in a structured way within databases for future use.

This research concludes that the eventual way of designing a more advanced and intelligent automated translation system is heavily depending on input such as: the formal language in which requirements are specified, the standardization of concepts in a domain specific language, and databases in which information and data in regards to client specific requirements has been captured. This research also contributed by concluding a set of preconditions for the automation of a more advanced and intelligent system: the operational functions such as Create, Read, Update and Delete (CRUD) need to be accommodated, the system needs to automatically store enriched words by means of a formal notation and standardized concepts, the system is required to automatically equate and allocate the enriched words on a system level, the system needs to automatically capture and distinguish definitions of the obtained words in relation to acting disciplines in order to formulate a product specification, the system needs to run on databases that contain valid knowledge obtained from a variety of projects as delivered in the past.

This page is intentionally left blank

Summary

The increasing complexity of the demand placed by clients in the construction industry nowadays, requires a different approach for contracting parties for the right translation from client specific requirements into product specifications. The increasing complexity of projects within the construction industry is also partly related to the amendments in obligations of contracting parties. These changes stem from the variety of integrated contract forms, in which contracting parties not only sign for manufacturing of design, but also for design and design management. Inexperienced clients are often not technically skilled. This makes it very difficult for this party to communicate the right performances of their desired product. There has been found in practice that this is mainly occurring in case of softer product requirements which are often qualitative from nature. This in contrast to the class of physical and functional requirements that are more quantitative.

A client's brief that consists of ambiguous demand specifications can often lead to misinterpretations for contracting parties. As a result, large deviations can arise in what clients expect, compared to what contracting parties think they have to deliver. This can have catastrophic consequences, assuming that clear agreements are missing during the early design stages about what is desired and what can be delivered. These consequences express their self in ambiguities during the design process that may result in the delivery of a defective product. These types of problems can often arise within processes in which product requirements are specified, translated and configured. There are various methods and techniques in use within the AEC-domain to capture client specific requirements systematically, efficiently and effectively as possible at an early stage. These strategies and tactics are introduced within these stages to reduce the changes of the revisions on the demand specification during later design stages.

This research focuses on the translation of client specific requirements into product specifications before and during the conceptual design phase. The objective of this research is to explore the possibility of introducing and implementing automation as a technique to optimize these processes with respect to efficiency. One of the main goals of this research is to optimize these translation procedures and to declare the level of ambiguity within client specific requirements. This especially in the case of soft product requirements, the non-functional requirements, which are often more qualitative from nature. These qualitative requirements are known to be difficult to interpret and to specify, contrary to quantitative requirements. This can partly be explained by the fact that there are no standards in what concepts within demand specifications actually imply, and how this relates in detail to design decisions at a system level. Ambiguity may result from this, which might reflect itself in an end product that deviates from the clients initial requirements.

Based on this problem definition, the main research question was formulated: "How can the translation process of non-functional requirements be structured and automated to formulate product specifications in the design process"? To formulate valid answers to this research question, a literature study was conducted on the design process, Systems engineering, Knowledge Management and Natural Language Constraints. The findings from the literature research were combined with the knowledge obtained from interviews that were held with specialists from the field of Systems Engineering. This with the aim to analyze how professionals deal with this problem in a practical environment. Based on the findings from these approaches, a prototypical program has been developed to test the knowledge.

The result of this research relates to a prototypical program. This program, on the one hand, accommodates a method that can translate both quantitative and qualitative client requirements into product

specifications. On the other hand, this program provides a digital environment in which words that are extracted from client requirements can be stored in a structured way for future use. The translation program, called 'The Bank of Knowledge', is intended to be used by designers and engineers as a decision support tool, to optimize the translation procedure of customer requirements into product specifications in terms of efficiency and effectiveness. The 'Database Manager Bank of Knowledge (BOK)' has been developed as a technique for specialized companies for the translation, structuring and storage of building information and data according to domain related terminology. This information and data can be integrated into a structured database that can be ingested by the 'The Bank of Knowledge' application.

This research has found that requirement translations into product specifications could be automated if the following input can be provided to feed computer based systems:

- 1) Validated interpretations of requirement with clients and users as obtained from previous projects;
- 2) Availability of a set of dissected requirements, as programmed in previous projects, that are represented in a measurable state;
- 3) The allocation of requirements and objects, as done in previous projects;

The eventual way of designing a more advanced and intelligent automated translation system is found as a byproduct of this research initiative. This, especially with the findings during the evolutionary prototyping process. These fundamental system requirements for such systems are summarized as follows:

- 1) Fundamental operational functions such as Create, Read, Update and Delete (CRUD), need to be accommodated within such systems;
- 2) Such systems need to be capable to run automated lexical analysis to dissect the linguistic chunks of text obtained from a client's brief;
- 3) Such systems need to be able to automatically store enriched words by means of a formal notation, as obtained from the dissected text, by means of standardized concepts to formulate the possible meaning(s) of the word within a sentence;
- 4) Such systems need to be able to automatically equate and allocate the enriched words in relation to the subsystems by means of a standard system distributions, such as the NL/SfB that is used within this prototyping attempt;
- 5) Such systems need to automatically capture and distinguish the translation of the definitions of the obtained words in relation to other acting disciplines, assuming that these definitions can vary;
- 6) Such systems need to automatically capture the final specification of the word(s) to formulate a product specification that satisfies the initial requirement;
- 7) Such systems need to run on databases that are filled with valid and state of the art knowledge obtained from a variety of projects as delivered in the past.

Samenvatting

De toenemende complexiteit van de vraag die door opdrachtgevers in de bouwindustrie wordt geplaatst, vergt hedendaags een andere benadering van opdrachtnemende partijen om de juiste vertaalslag van klantwensen naar productspecificaties te kunnen maken. De toenemende complexiteit van projecten binnen de bouwnijverheid hangt ook deels samen met de hervormingen in verplichtingen van opdrachtnemende partijen. Deze wijzigingen komen voort uit de verscheidenheid aan geïntegreerde contractvormen waarbij opdrachtnemende partijen niet alleen tekenen voor het ontwerp zelf, maar ook voor het ontwerpproces en de procesbeheersing daarvan. Doordat onervaren opdrachtgevende partijen vaak niet technisch onderlegd zijn, kunnen er problemen ontstaan tijdens de communicatie van de juiste specificaties van het door hen gewenste product. Dit is een fenomeen dat zich vooral voordoet in de groep van de zachtere producteisen, in tegenstelling tot de klasse van de eisen die numeriek uit te drukken zijn.

Een onvolledig gespecificeerde vraag kan vaak leiden tot misinterpretaties bij de opdrachtnemende partijen. Hierdoor kunnen grote afwijkingen ontstaan in wat opdrachtgevers willen, vergeleken met wat opdrachtnemende partijen denken te moeten leveren. Als er niet gedurende de initiële ontwerpstadia duidelijke overeenstemmingen geformuleerd worden over wat gewenst is en wat geleverd kan worden dan kan dit catastrofale gevolgen hebben. Deze gevolgen uiteten zich in onduidelijkheden gedurende het ontwerpproces die mogelijk leiden in een gebrekkig opgeleverd product. Problemen kunnen hierdoor vaak ontstaan binnen processen waarin product eisen gespecificeerd, vertaald en geconfigureerd worden. Er worden binnen de bouwindustrie vele methoden en technieken toegepast om de wensen van de klant vroegtijdig zo systematisch, efficiënt en effectief mogelijk vast te leggen. Dit wordt gedaan zodat er later geen revisies doorgevoerd hoeven te worden in de programmering van de daadwerkelijke vraag tijdens de verschillende ontwerpfasen.

Dit onderzoek richt zich op de vertaalslag van klantwensen naar productspecificaties voorgeand en tijdens de schets ontwerpfase. Dit met als doel automatisering te introduceren als techniek om deze processen te optimaliseren ten aanzien van efficiency. Eén van de voornaamste doelen van dit onderzoek is om deze vertaalslag te optimaliseren ten aanzien van eenduidigheid. Dit voornamelijk voor de zachtere eisen die niet direct numeriek uit te drukken zijn. Deze kwalitatieve eisen zijn in tegenstelling tot kwantitatieve eisen lastiger te interpreteren en te specificeren. Dit kan deels verklaard worden doordat er geen gestandaardiseerde opvattingen zijn over wat voorkomende concepten binnen vraagspecificaties daadwerkelijk impliceren, en hoe dit zich gedetailleerd kan verhouden tot ontwerpbesluiten op systeem-niveau. Hieruit kan ambiguïteit voortvloeien waardoor de wens van de opdrachtgever (wellicht) niet gerealiseerd zal worden.

Op basis van deze probleemstelling is de hoofdonderzoeksvraag opgesteld: “Kan de vertaalslag van niet-functionele eisen zo worden gestructureerd en geautomatiseerd, dat er productspecificaties geformuleerd kunnen worden binnen het ontwerpproces”? Om geldige antwoorden op deze onderzoeksvraag te formuleren is er een literatuurstudie uitgevoerd over het ontwerpproces, Systems engineering, Kennis Management en Randvoorwaarden uit natuurlijke taal. De bevindingen vanuit het literatuur onderzoek zijn gecombineerd met de kennis die is gevonden vanuit interviews die zijn gehouden met specialisten vanuit het domein van Systems Engineering. Dit met als doel om meetbaar te maken hoe professionals omgaan met deze problematiek in een praktische omgeving. Op basis van de bevindingen in deze benaderingen is er een prototypisch programma ontwikkeld om de kennis te testen.

Het resultaat van dit onderzoek verhoudt zich tot een prototypisch programma. Dit programma accommodeert enerzijds een methodiek die de vertaalslag van zowel kwantitatieve als kwalitatieve klanteisen kan vertalen in productspecificaties. Anderzijds levert dit programma een digitale omgeving waarmee woorden die onttrokken zijn vanuit klanteisen gestructureerd vastgelegd en opgeslagen kunnen worden voor toekomstig gebruik. Het vertaalde programma, genaamd 'The Bank of Knowledge' (De Bank van Kennis), is bedoeld om ingezet te worden door ontwerpers als ondersteunende techniek om de vertaalslag van klanteisen in product specificaties zo efficiënt en effectief mogelijk te maken. Daarbij is de 'Database Manager Bank of Knowledge (BOK)' ontwikkeld als techniek voor gespecialiseerde bedrijven voor de vertaling, structurering en opslag van informatie en data over domein gerelateerde terminologie. Deze informatie en data kan geïntegreerd worden in een gestructureerde database die gekoppeld kan worden aan 'The Bank of Knowledge' applicatie.

Dit onderzoek heeft bevonden dat de vertaalslag van specifieke klant eisen in product specificaties gestructureerd en geautomatiseerd kan worden indien een programma beschikt over de volgende input:

- 1) Gevalideerde interpretaties van klanten en gebruikers met betrekking tot concepten
- 2) De beschikbaarheid van een ontlede set aan eisen die zijn vertaald in meetbare gegevens
- 3) De toekenning van eisen aan objecten binnen een gebouw ontwerp (BIM model)

Dit onderzoek toont ook aan dat de ontwikkeling van een intelligent geavanceerd geautomatiseerd vertaling system gerealiseerd kan worden indien aan de volgende eisen voldaan kan worden:

- 1) Fundamentele operationele functies zoals de Creatie, Inlezen, Wijzigen en Verwijderen van informatie en data geacommodeerd zijn in het programma
- 2) Tekst geautomatiseerd ingelezen en ontleed kan worden
- 3) Woorden vanuit de zinnen herkent en verrijkt worden door middel van gestandaardiseerde concepten die geautomatiseerd vast gelegd kunnen worden met behulp van een formele notatie in een database
- 4) Verrijkte woorden vanuit een specifieke eis geautomatiseerd toegekend kunnen worden aan een gebouwonderdeel door middel van een specifieke gebouw classificatie methode
- 5) De (meervoudige) betekenis van woorden vanuit een specifieke eis onderscheiden, vertaald en toegekend kan worden aan actoren
- 6) De daadwerkelijke definitie van de woorden vastgelegd kan worden om zodoende een product specificatie te kunnen formuleren die de initiële eis bevredigd
- 7) Een database geïntegreerd is die bestaat uit betrouwbare informatie waarop het systeem rust

2 Introduction

The complexity of the projects within the architecture, engineering, and construction industry (AEC-industry) is increasing rapidly in this modern era. This increased technological degree where the contractors, Designers and Engineers (D&E) are faced with, can (partly) be explained by the variety of predefined building requirement as a function of the demand specification. The difficulty of the project within the AEC-industry has also increased over the years due to changes in the responsibilities and liabilities of the parties involved (Chao-Duivis, 2017). Their participation within the projects commands a methodical justification of the demand specification as specified by the clients (Lenferink, Tillema, & Arts 2013). Translating the variety of requirements into product specifications is a complex task, especially if little information is available and numerous aspects need to be reconsidered during the early design stages. The impact of design decisions taken within the earliest design stages affect the product its final configuration the most (Aliakseyeu, 2003). This complexity induces the AEC-industry of an integral way of working, more than ever, to adapt and react to the market demand as early as possible. This approach must lead on one hand to a possible reduction of time during design processes, and an increase of quality by product design on the other hand (Abanda, Zhou, Tah, & Cheung, 2013). Coping with such market conditions is from great importance, especially when the contractor is responsible and liable for both product design and its manufacturing. This form of contracting yields from test projects that are initiated and executed under the UAC-IC 2005 by the realization of Civil works within the Dutch construction industry (Chao-Duivis, 2017).

To attain this higher quality in both product and process design, processual strategies and techniques might need to be adjusted to harmonize and safeguard process and product performances. The information in requirements interacts greatly within the verification processes. Within the verification processes, the design is tested on the compliance with the initial requirements as stated within the initial demand specification. These processes are very time consuming and failure sensitive. This failure sensitivity gets emphasized especially if numerous ambiguous requirements have been specified by the client. The employee needs to be able to justify the processual steps that are taken during the translation procedure of linguistic chunks of text (the requirement) into product design (specifications). This might be an achievable task in scenarios where unambiguous requirements have been contracted. However, this translation procedure can get very critical when non-functional requirements are contracted. This, especially when the contractor has little or few experience with aspects such as the type of product, design, design management and systems engineering. Within this research, there will be studied upon the characteristics of client specific requirements in order to relate the knowledge that is required for the translations into product specifications.

Therefore, the research of user and client interaction, requirement management and (automated) constraint checking of building information has come forward. This fields of research mostly aims for a reduction of project variance on one hand, and to increase a higher product quality at the

other hand. The process of user client interaction, requirement management and rule checking is researched upon greatly given the current market conditions. Client demands are nowadays often known to be complex from nature due to the high product performances that these parties aim for. This might lead to problems during design and manufacturing given the fact that not all designers and manufacturers are experienced with certain demands. The profits that contractors are gaining from moderate projects is not that high. This forces contractors to pay attention to the development of these aspects given the risks that they are taking by contracting complex demand specifications where they are unfamiliar with. There is a clear borderline between taking risks as a contractor or a gamble. Within this research, there shall be researched upon the possibility to introduce a certain Design and Decision Support System (DDSS) for the translation processes for client specific requirements as provided by unprofessional clients in demand specifications.

2.1 Motivation

Managing the compliance and performances of client specific requirements has grown with the introduction of integrated contracts (UAC-IC) within the building industry. This form of contracting is known to be used for contracting Civil works within the Dutch construction industry since 2000 (Chao-Duivis, 2017). There have been numerous (test) projects, mostly Civil works, that are contracted according to this contract from between the year 2000 and 2017. These type of contracts have resulted in an alteration of responsibilities and liabilities within design processes (Lenferink et al., 2013). This shift has contributed to a higher responsibility of the contractor (employee) during the overall construction process. Despite the traditional way of working of contractor (built as designed), this type of contracts force the contractor for transparency of the performance of a design both during the design stages as in the usage phase. There might be assume that having a design that fully complies with the requirements before a building is manufactured, might yield in a decrease of the failure costs and improvement of the quality for both the client and the end user (Moonen, 2016).

The management of the client specific requirements remains very difficult given the fact that a client might not know exactly what he wants at the very beginning of a project (Moonen, 2016). This might yield in ambiguous defined product requirements that are hard to translate into product specifications on a system level (the building). Defining a valid model before it is realized might therefor be a very difficult task since clients are known to define their need iteratively by evaluating and updating their requirements rather than narrowing them down initially (Kim, Kim, Cha, & Fischer 2015). Therefore, contractors should be adaptable to this iterative character and manage the requirements in a very sophisticated way (Moonen, 2016). A contractor should be able to determine the ranges in which design decisions should be taken. These ranges state the bandwidth of decisions that other designing parties that are involved should be taken. These ranges reflect the specific goals that contribute to the performance of their products in compliance with the initial requirements. Currently, the management of requirements remains a manual process. This process requires a lot of investigation on the information about requirements and the design (Moonen,

2016). These are very critical and laborious processes. The gained information from the projects as executed in the past might be stored in databases. However, the usability of this type of information relies on the data structure, the technique in which the data is stored and how users can possibly consult the specific data. The information from these databases require a lot of effort by converting them to useable input on their own. Investigating this area is there for a sport in its own. Researching how this information should be documented, enriched and stored is there for from great importance in both its practical as scientific contribution. The usage of this information and data for the translation of requirements into product specification will improve the efficiency and quality during successive design and manufacturing stages.

Predictions on the configuration of subsystems within the earliest design stages, as derived from the demand specification, seems to be a very complex task. This complexity increases especially by scarce ambiguous information and data on product requirements as provided by the unprofessional client. The constraints that result from these linguistic chunks of text affect the product configuration. It is a complex task to derive the right information from requirements, especially if they are formulated in an ambiguous state. A designing party has a lot to do with constraints. Within all these constraints there will be approximately 15% of freedom, the other 85% should translate the client's needs (Gehry, 2017). The AEC-domain is lagging behind of implementing methodologies that promote logging and storage building information according to unambiguous standards as a function of time (CB-NL, 2015). These methodologies might be introduced and dictated by clients, but are not safeguarded on a practical level by the contractors. This events might provoke complex court cases.

Using building information and data as generated from previous projects might promote the unfavorable conditions in which design and manufacturing's decisions are taken within the AEC-domain. Dutch organizations, such as the Instituut van Bouwrecht (IBR) or the Centrum voor Regelgeving en Onderzoek in de Grond-, Water- en wegenbouw en verkeersvoorzieningen (CROW) that are concerned with the problems that correspond with the translation procedures of non-functional requirements haven't found a formal standardized way in which these can be treated. However, there needs to be mentioned that these institutions are providing guidelines as a support tool. Still, these are not mandatory to use in practice. Whenever these defaults within the translation processes are analyzed in both a practical as scientific way; then statements about automation can be formulated in regards to developments of (semi) automated tooling.

Thus, the complex set of requirements that are included in the demand specification as presented by the unprofessional clients might be causing the urged need for expert systems. These systems could possibly affect the design stages positively. The world counts an almost infinite amount of real estate wherefrom its information is lost. This is a pity since this type of building information can be re-used for numerous applications within the field of design and decision making. These systems could be filled with these type of building information and data. These tools can be introduced as design and decision support tool by product design within the early design stages

where little information and data is available. Within these early design stages, certain experts systems (ES) or knowledge based system (KBS) can be consulted for queries on the translation processes as logged from the past. The introduction of such tools could clarify and discard design contradictions to reduce the risks and defaults in successive processes. The current techniques that are used within the AEC domain during the translation of product requirements into product specifications are executed manually rather than automated (Niemeijer, 2011).

2.2 Problem definition

The interpretation and translation of client specific requirements into product specifications prior and during early design stages of complex building projects is a very critical phase. Programming a design with the initial requirements, as stated from the demand specification, can be a laborious task. The eventual conformity towards client specific requirements is of great importance to safeguard the functionality and the performance of a product. If a contractor remains in default by non-compliance towards these requirements he might be held liable and fined. These costs mostly correspond with extensive additional project costs and extended project durations. Both of these negative events might harm the integrity of a contractor within his field of business. There are numerous court cases held on these issues where the client has not got delivered, in his or her experience, of what he or she asked for.

The fact that there are no standards, only guidelines, in which requirements have to be specified by unprofessional clients might make the journey to unambiguity in demand specifications even tougher. This affects the complexity by managing the compliance towards requirements in the design process, especially by non-functional requirements. Here, within this research, we assume the following definition of an unprofessional client: a client that is untrained, unfamiliar, and unqualified for programming and governance of the design and manufacturing processes that contribute to product delivery". This definition implies that the client's liability and obligations are only rooted in the development of the client's brief which result in the demand specification.

There is no formal method in which non-functional requirements can be interpreted, translated and specified. This offers the opportunity to improve this process within the design phase. This can be promoted by means of an exploration towards a methodology and technique that can accommodate this need. There are currently no databases which can be consulted for queries on the translation of non-functional requirements. These are not available in companies, nor in literature nor on the World Wide Web. These techniques might be beneficial for the design process by defining this information only once, capturing its enriched form, and translating this in to reusable information. These systems could create managerial edges since knowledge gets produced, captured and evaluated.

The main problem that is investigated in this research can be found in Building Information Management. More specific, Building Information Management during early design stages where translations are executed from client specific requirements obtained from clients briefs into to product specifications. These procedures are currently known to be executed manually, to be time

consuming, and error prone. These processes are assumed to be very profitable on the long run if executed accurately, given the simple fact that tenders can be won if products are configured within the holistic ranges as client's desire. A lot of knowledge is required for this type of decision making by uncertainty and a scarcity of accurate information. Capturing the specific knowledge in a knowledge based systems isn't happening in the AEC industry yet. This makes a specific demand a unique design task each time.

2.3 Research scope

The research scope will be demarcated within this section to clarify the specific intention of this research initiative in regards to both its practical and scientific contribution. This is from great importance to prove the effectiveness and efficiency of automation by both electronic requirement management and translation procedures. This, especially for the purpose of knowledge capturing as a function of time.

This research focuses on the translation of non-functional requirements obtained from the client's brief. The class of physical and functional requirements will be introduced to demonstrate how this translation procedure differs from the class of non-functional requirements. Here, a case study will be introduced wherefrom a case model will be derived. This scope has been chosen for evaluation as there is a vast amount of these types of requirements that are contracted which are not integrated or even treated by building design. This might make contracting an even risk fuller business than it currently is.

Non-functional requirements are also the type of product requirements that might require a holistic approach by programming their specifications. A room, for instance, can be configured by requirements that need to be specified from the disciplines such as architecture, structural-, electrical-, mechanical engineering and building physics. More specific, requirements from all parties involved within a designing line-up might contribute to the satisfaction of a certain non-functional requirements at once.

If we take a closer look to the following non-functional requirement: *"The room needs to withhold a cozy atmosphere"*; it might be though to derive which parties needs to take which actions. This process could be a very complex and laborious task. Within this research project we analyze the impact of the architectural and building physical requirements in regards to the configuration of the product specification. Here, there is assumed that if a single requirement can be distilled from a non-functional requirement; the rest will follow as it is a matter of time. Distilling all product specifications from a non-functional requirements in regards to all disciplines involved, in an unambiguous form, is what there will be aimed for in the ideal situation.

This approach gives a good opportunity to evaluate the possibilities of using automation by requirement translation processes. Within these processes there will be checked and tested how automation can be introduced to promote efficiency and effectiveness to achieve greater and unambiguous product specifications. This prototype is evaluated with a use case to define the

usability, efficiency, effectiveness and limitations of using automation for non-functional requirements translation processes.

2.4 Importance

Research upon the possibility of semi-automated requirement translation is from great importance as this can improve the early design process and the overall quality of a final product. This process can discard ambiguity by the interpretation, translation, and specification of client specific requirements which might prevent costly mistakes. Specifications are supposed to formulate a final performance of a certain object within the building information model, yet, there is known that not all specifications are deduced from requirements during early stages. This is often affecting the performance of a certain element within a building. This implies that the desired product performances are not covered by design. Clients want to invest in products in which performances can be justified according to their initial demand. Things can go (badly) wrong for D&E, in terms of finances and integrity, when decisions cannot be clarified during design stages and product delivery.

Besides from building information management, the possibilities for automation of certain design and manufacturing processes are of great importance to research upon as this might improve the design process. The introduction of automation can discard a lot of repetitive administrative work which will be minimized. Then, the focus can be brought upon other business process from a more profitable nature. The investigation on automation for requirement translation procedures seems a useful topic where business and quality improvement, assurance, and building information management fuse. Especially, given the fact that there are no such systems in practice within the AEC-domain, or offered by market, that accommodate these functions.

2.5 Related work

This research initiative is driven by the intention to contribute to the required knowledge for the development of a prototypical (semi-automated) information system in which requirements can be interpreted, translated, allocated, and specified during early design stages. This, with the assumption of uncertainty by decision making due to the scarcity of building information in early design stages. With this as given, there has been reviewed by means of a literature review and in-house practices, on what previous researchers and practitioners have contributed to this domain. The review of literature revealed the current status on which such techniques are introduced in both the AEC domain and other industries. The in-house practices were used as a mean to investigate, on a more practical level, to which extend experts from the field are familiar with such techniques and information systems. The merged outcome of these two processes provided the fundamental insight in the state of the art methods and techniques regarding this research problem. There has been found that such information systems are not existing, both for the in AEC-domain and other industries, in which requirements can be interpreted, translated, allocated and specified. However, various methods and techniques as used in other industries could contribute to the fundamental knowledge that is required for the development of such an information system. To be more specific, several interesting methods and techniques in regards to the design process,

legal and governance, requirement management and engineering, linguistics, and data & text mining have contributed to the required knowledge for the development of the prototype. This implies the possibility to develop such a specific system by means of merging the knowledge due to the findings as obtained from those processes.

Niemeijer (2011) contributed by his work on constraint specification in architecture. He identified the variety of methods and techniques by the application of constraints as written in natural language within the AEC-domain. He also revealed how both humans and computers could cope with chunks of text as written in natural language. Niemeijer (2011) treated the relation between client requirements and constraint specification for information systems within the AEC, which was from great importance for this research. This due to the fact that a requirement can be assumed to be compelled in constraints; and that a constraint is formulating a certain decision bandwidth. However, his work emphasized constraint checking rather than constraint solving. In this research we are fundamentally trying to solve a constraint which a requirement dictates, rather than checking it afterwards. One can state that we are basically doing both, by solving the constraints after translation in specification(s); and checking constraints during verification. One way or another, Niemeijer (2011) contributed a lot by providing the state of the art knowledge in regards to constraint specification in natural language for information systems within the AEC industry.

The fact is that there is aimed to shell a methodology in which requirements can be interpreted, translated, allocated, specified, and stored as a function of a client demand to improve client / designer interaction in early design stages. Therefore, a variety of traditional requirement management methods as used in various (designing) industries, which are often not mentioned within this report, have been reviewed upon in a practical setting to determine how the best of all these approaches could be merged for use in such a (semi-automated) information system. There has been reviewed on how the generic theorems are treating these translation procedures. Jallow et. al (2017) published a paper in which they presented a first attempt for the development of 'An enterprise architecture framework for electronic requirement information management (eRIM)' within the construction industry. Their work covered the fundamentals of the concept of requirements management, traditional conventional requirement management models, and how these both can be linked to Enterprise architecture (EA). Jallow et. al (2017) confirms the urged need of such an information system, as aimed for in this research, by the following statement: *"Little research has advanced in requirements management in construction, and no known development has been reported in the use of specialized software for requirements in construction. Even where a system exists, an underlying framework must be available to specify how that system should be used, factoring in the lifecycle phases & processes, information structure, information flow within the organization, and the process for managing changes."* Jallow et. al (2017) contributed a lot by their work in terms of knowledge and development to this domain, however, even the eRIM system is not accommodating the functionality in which client specific requirements can be interpreted, translated, allocated, specified, and stored.

2.6 Primary hypothesis & objective

In regards to the problems that occur for the translation of client specific requirements into product specification, a prototypical information system will be developed which is user friendly to the layman. The hypothesis of this research states that an automated translation tool will improve the early design process. This with the assumption that the system is providing valid knowledge for the translation of requirements into product specifications. The objective for the development of the tool is to research whether valid knowledge from the past can contribute to the accuracy of decision making for design. Such tool could support the determination process of ranges in which design decisions should be taken. This development is initiated to overcome the time consuming, failure sensitive manual process during early design to solve problems that are occurring during current procedures. This could deliver a prototype which is based upon automated (web based) lexical analysis of client specific requirements, (web based) word enrichment by state of the art knowledge, word to (sub)system allocation, and product specification. This tool could capture and store the obtained knowledge in its database, and be able to automatically expand as a function of time. The captured knowledge within the system its database will grow, which implies that it shall provide more and more knowledge during its reference period. This implies the usage of such information system for other use cases as well, these will be discussed later this report.

2.7 Research questions

This section treats the research questions. These research questions are derived from the problem definition and research scope. Within this research project, the main research question is:

“How can the translation process of non-functional requirements be structured and automated to formulate product specifications in the design process?”

The main research question is divided into 6 sub questions to gain specific knowledge. These sub questions are closely related to parts of the research design. These parts are the design process, knowledge management and natural language constraint in Architecture, Engineering and Construction (AEC).

The design process is evaluated with the aim to gain knowledge on Systems Engineering, verification of requirements and the information exchange which is happening in the process. Knowledge management is studied upon to measure the current state in which procedures are standardized and formalized for capturing and storing data, information, and knowledge. Natural language constraints in AEC are treated with a view to gain insight in the current state of the methods and techniques in use for translating and processing chunks of fuzzy text into Domain Specific Language (DSL). This is done to evaluate the possibilities to create an automated knowledge based system that can be consulted by the translation of non-functional client specific requirements. These three subjects lead to the following 6 sub questions:

- 1) What client specification procedures are there in use within the design process, and how does Systems Engineering support these procedures?

- 2) What variety of client requirement types are known within the design process, and which of these carry risk in terms of non-conformity?
- 3) What is the current practice in the AEC industry for translating client specific requirements into product specification, and how do verification procedures safeguard these?
- 4) What can automation, for translating client requirements into product specifications, contribute to the design process?
- 5) What are the current techniques within the AEC-domain, by means of automation, to translate product requirements into product specifications?
- 6) Is it possible to develop a method that translates and stores physical, functional, and non-functional requirements into product specifications by means of automation?

2.8 Research design

This research is structured by 3 parts that are closely related to the research questions. Figure 1 visualizes the path that this research project shall follow. All of the research question as mentioned in the previous section are researched upon in two ways. Firstly by means of existing literature and secondly with an evaluation of the current practice by interviewing experts from the industry. The first part requires a theoretical research that will be executed. Here, a literature review will be conducted on the main themes known as the design process, knowledge management and natural language constraint in Architecture, Engineering and Construction (AEC).

The focus will rely on the use of Systems Engineering, the verification process, and requirement management within the literature review of the design process. After the theoretical research, a qualitative research upon the current practice will be executed. Interviews with experts that are coping with the defined problems will be held to evaluate the design process and evaluate the processes where (major) errors are occurring from. These findings will be used to evaluate how automation can possibly counteract and support these processes. The same procedure will be used for the second and third part of this research project. Within these parts, the current practices on knowledge management within the AEC-industry along with natural language constraint in the AEC-domain will be investigated. The conclusions of the interviews, along with the evaluation of the literature, will be translated into the scope for the development of the prototypical 'requirements translator' and 'database manager'.

To evaluate the possibilities for the development of a prototypical system in which requirements can be translated, specified and stored, a selection in requirements will need to be made to define the scope of this research. This will need to be done according to the various types of requirements. These types are identified with an analysis of requirements databases as gained from existing research projects. The applicability of the prototype will focus on the translation and capturing of physical, functional, and non-functional requirements. The translation of (ambiguous) non-functional requirements has received little attention in research. The following figure illustrates the research model:

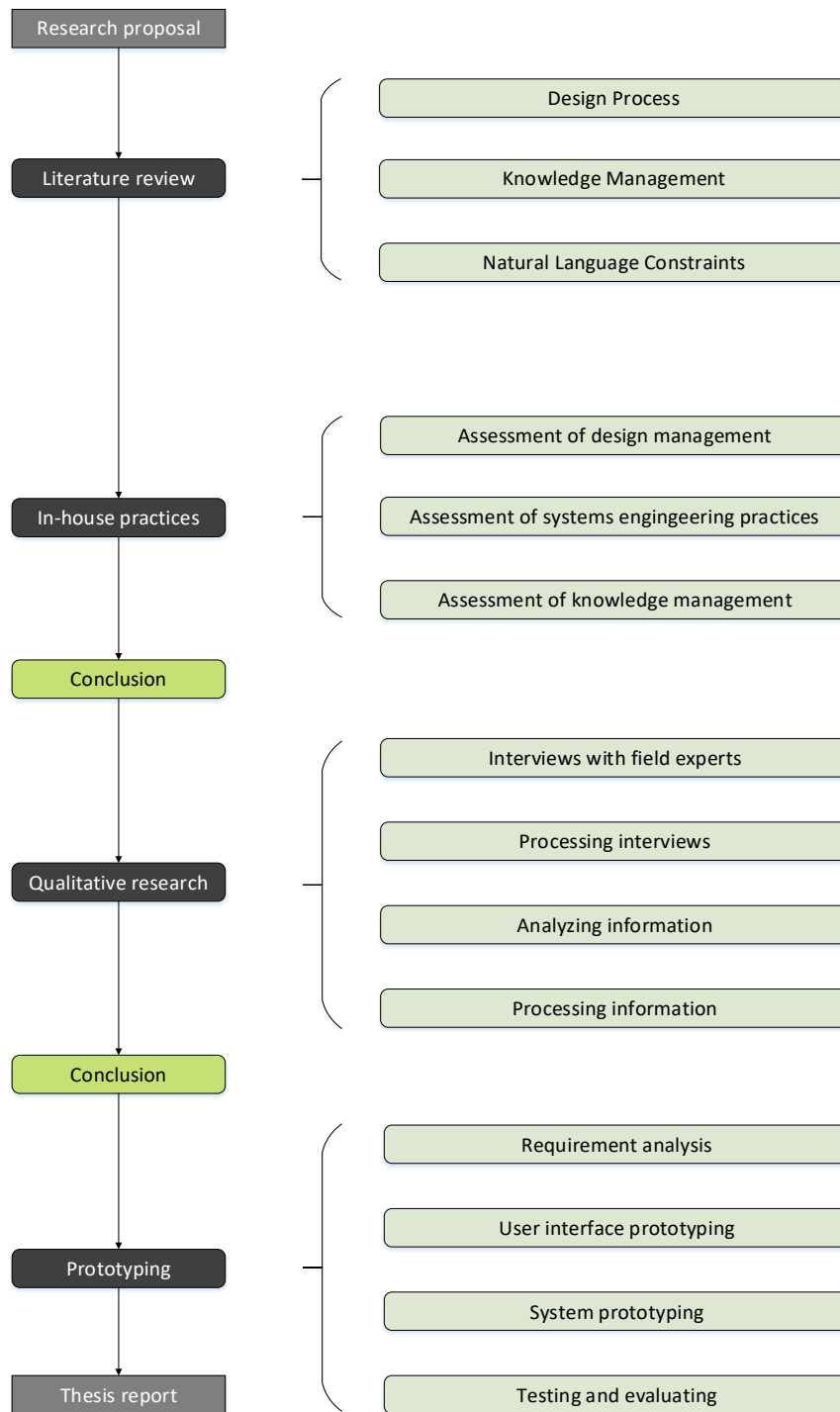


Figure 1: Research model.

2.9 Expected results

The results as expected from this research shall be treated within this chapter. The expected results of this research project will come in threefold. The first part consists of a literature review upon the design process, knowledge management and natural language constraint in Architecture, Engineering and Construction (AEC). Hereafter, a clear descriptive evaluation on the current findings will be given to demark the scope and the necessity for a methodology or system that treats the translation of non-functional requirements. Here, the applicability and opportunities will be analyzed and captured. This literature review can be found in chapter 3.

In the second part of this research project, interviews will be held with field experts. The outcome of these interviews will be used to evaluate on problems by translation and verification of non-functional requirements, and where these are occurring within the design process. The evaluation of these interviews will be put in a report outside of this thesis report due to confidentiality obligations. The main findings will be summarized in chapter 4. The outline for automation by translation procedures for non-functional requirements will be given with the purpose to prototype a certain system. In the 5th chapter this outline and scope will be used for the development of the prototypical requirement translator. This program is assumed to be fed with the use of valid information and data gained from both literature and the World Wide Web. This is used to ground the systems reasoning. Sources that might be consulted within these processes are norms, standards, dictionaries and linguistic methods and techniques. The purpose of the system is to execute a (semi) automated activity in which non-functional requirements can be checked by means of automation on their lexical composition, enriched with useful information and data, assigned to subsystems within the system and be converted into specific disciplinary product specifications. The system is fundamentally breaking down non-functional requirements into raw engineering's data where design and manufacturing's decisions can rely upon.

The system needs to provide the possibility in which knowledge can be created, captured, evaluated and reused. The system is aimed to be able to create, read, update and delete (CRUD) information and data as a function of time. These type of systems might get smarter and more convenient to use as a function of its usage period since it can be thought to think and learn on its own also. Within this research initiative, we aim to place the first building block of such a prototypical system. Numerous use cases can be stated by use of these type of system besides this research project its initial intention. This will be discussed in the recommendations section of this report. The system relies upon the definition of concepts which are gained from literature and upon the knowledge which is built up in the requirement translator its database by usage. The fusion of all subsystems within such a translation system will form the final prototypical expert system where future IT-developments can possibly rely on. The development of this prototypical system can be found in chapter 5 and beyond.

This page is intentionally left blank

3 Literature review

3.1 Motivation

This literature review is held to explore and evaluate the actual knowledge that recent research contributed to this research project its problem definition. This method provides a systematical approach that will be used to review actual research and developments in relation to this research initiative. In the literature review an overview is given in various topics that are assumed to be closely related to the origin of this research project its problem definition.

The first topic is the design process which is evaluated to define the scenarios and process where client specific requirements are used in. This procedure can be defined as the process where the actual client specific requirements are functioning as the information and data that will be used as the input of product or process design. The focus will rely on product requirements in this investigation. Analyzing this process is required in order to evaluate the possibility of using automated requirement translators within the design stages of building design.

The second part of this review on literature covers the domain of knowledge management. Here, knowledge management will be reviewed on its relation to the AEC industry. This chapter will treat the fundamental explanation of knowledge management, how this can be managed, its goals, and applications. This is knowledge is from great importance for software prototyping.

The meaning and application of Natural language constraints within the Architecture, Engineering and Construction industry is the third topic that shall be treated. Here, constraints as products obtained from natural language will be analyzed in relation to their application for both architectural design and as input for information systems. The evaluation of this literature review reveals the possibilities for the design and development of an information system that can be consulted for the translation of fuzzy requirements into product specifications during early design stages.

3.2 Design process

Within this section, the development of a design during a construction project is evaluated to define the information and data development needed for verification procedures. This is a very critical process, especially for the verification of (ambiguous) fuzzy requirements since these group of client specific requirements are not directly measureable (Moonen, 2016). The design process is defined in different phases of a project life cycle in a building project within the AEC-domain. According to BIMforum, there are various definitions of standards for the processual structure of design phases. Given this assumptions, there needs to be mentioned that this research will focus on the Dutch construction industry.

The design of the system, the building model, can be a complex task due to the collaborative environment of the AEC domain. This collaborative environment expresses itself in to the multi-disciplinary line-up that is collaborating as a function of an end product; the building. Since its introduction over about 50 years ago during the late 1960's by Brunton et al. (1964), the concept of Architectural Management remains open to interpretation in the literature. This despite numerous studies that have articulated the importance of adopting such concept, especially by the CIB Working Group w096 Architectural Management (Emmitt et al., 2009). CIB W096 is the only international network dedicated to examine and promoting AM (Alharbi, Emmitt, Domain., 2015). This group has yet to adopt a final definition of this concept which is a criticism that can be made of their only book named as Architectural Management: International Research & Practice (Emmitt et al., 2009). With this as a given, this research adopts the following original and recent definition of AM, which is grounded by empirical research: *"Architectural Management (AM) is the strategic management of the architectural firm that assures the effective integration between managing the business aspects of the office with its individual projects in order to design and deliver the best value to all stakeholders (Alharbi, 2015)"*. According to Alharbi et al. (2015), Architectural Management dissected into two distinct parts, given: office or practice management and project management. The former provides an overall framework within which many individual projects will be recommenced, managed and completed (Alharbi, Emmitt, Demain., 2015). Both parts have the same objectives, but the techniques vary and mesh only at certain points (Brunton et al., 1964).

The poor management of early design phases has proven to be the cause for document defaults and rework (El. Reifi & Emmitt, 2013; Tilley, 2005). It is in the early stages of the design phases where the influences of stakeholders is largest and the costs of changes are lowest, making this the best stage for value realization (Samset, 2008). The definition of phases within the Dutch construction industry is defined by the Dutch standardization institute in the Dutch standards (NEN, de NEderlandsde Norm) and the definition of "The New Rules" (DNR, De Nieuwe Regeling). These standards and definitions are created by Dutch institutional organs such as the NLIngenieurs and BNA. The DNR-STB & NEN2574 define then phases of a construction project (BNA, NLIngenieurs, & ONRI, 2009; Nederlands Normalisatie-instituut, 1993). The DNR-STB and NEN2574 are defined for the use of traditional contracts. In this traditional form of contracting, a tender shall be put in the market whenever a design is finished. The manner of the timing of pricing and tenders vary greatly

amongst the different types of contract forms that are in use within the Dutch AEC-industry (Chao-Duivis, Koning, & Ubing, 2013). This section of the literature review is mainly focusing on the design process rather than the pricing phase. Therefore, the pricing phase is not evaluated within this overview. The schematic representation of the design phases has been merged and represented in Figure 2.

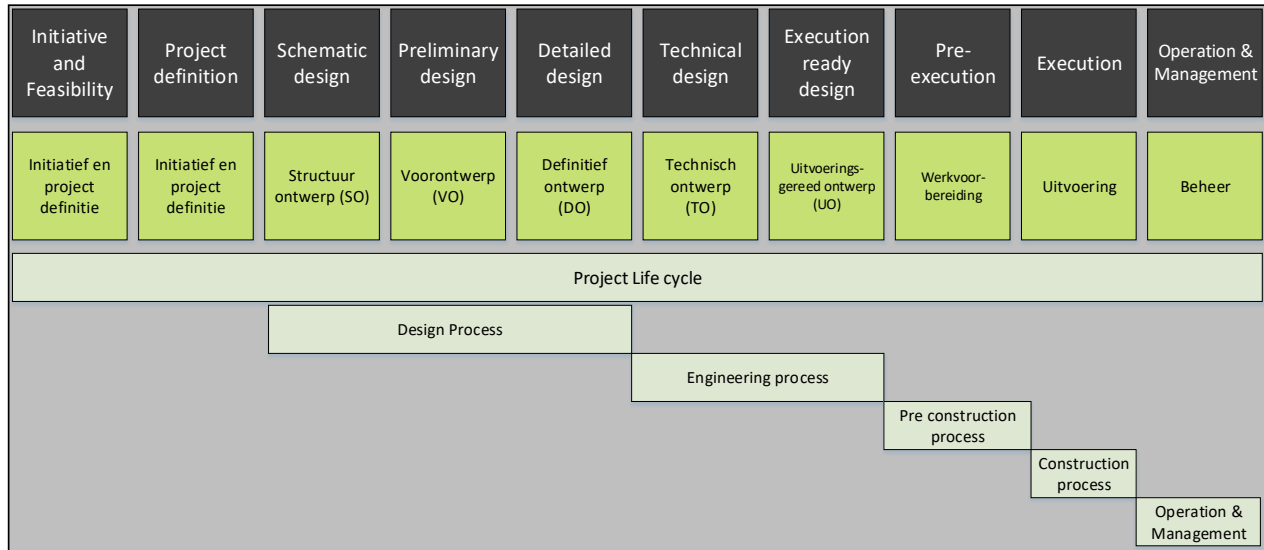


Figure 2: Project life cycle of a construction project with the project phases based upon BNA et al., 2009; Eadie, Browne, Odeyinka, Mckeown, & McNiff, 2013; Nederlands Normalisatie-instituut, 1993.

According to Moonen 2016, the development of information and data within these phases can be dissected in two different groups. The first group can be classified as the raw information of requirements which is provided by the clients in charge (Moonen, 2016). The second group can be classified as the information which is created in the design which reacts to the requirements (Moonen, 2016). The design should be generated according to the client's requirement and the environment the project is programmed in. Here, we find that the client specific requirements and the environmental conditions are formulating the fundamental constraints. These constraints will be specified as the ranges in which design decisions need to be taken. The interaction between the different sources of information can be found within these processes. The early stages within a design process have a more conceptual and iterative character. This can partly be explained by the scarcity of both design and engineering's information and data available at this point. The design phase can be characterized by a top down approach (Moonen, 2016). Here, design decisions are made about the key elements of design in the earlier phases. This approach can partly lead to a more linear process eventually where decisions are developed in a technical way in subsequent phases. The development of a design in relation to its iterative character is visualized in Figure 3.

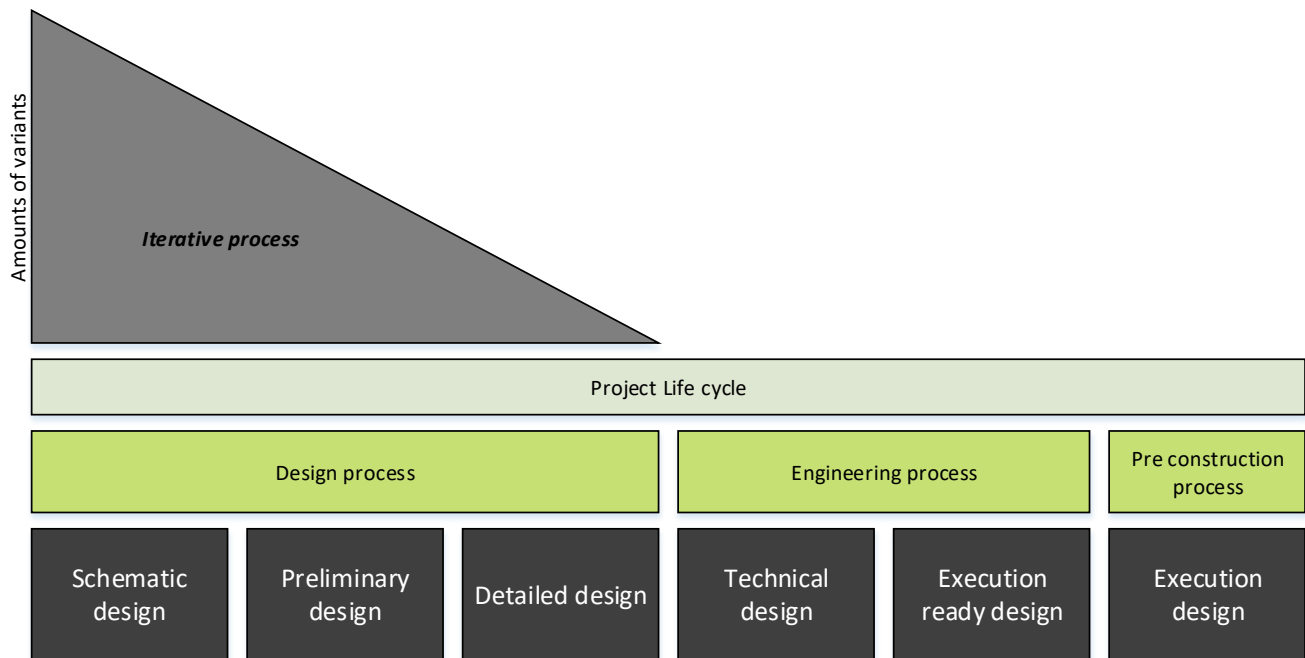


Figure 3: Iterative character during the design phase (Moonen, 2016)

According to Moonen 2016, the biggest decisions are made within the early design phases. Here, decision upon the core concepts of a design will be captured. Numerous amounts of variants are produced within these stages since the ultimate criterion for the design hasn't been defined yet. There is an inversely proportional relationship between the amount of variants as well as the impact of decisions in regards to the final project definition. The further the design process proceeds, the fewer the variants and the impact of the decisions will be and the more specific the project definition shall get. This phenomenon can be explained by the fact that the cost of changes will increase when changes are made in later phases (Lu, Fung, Peng, Liang, & Rowlinson, 2014). This principle its effect has been included in the Macleamy Curve and visualized in Figure 4. The elaboration of a design will be of a higher level at later phases, this is one of the main reasons why the costs will rise for revisions in the design. Possible reconsiderations instigate numerous extra process costs that shall manifest their self in subsequent phases of the project. This is one of the biggest reasons why the amount of variants should be declining as a function of time in the design process (Moonen, 2016). With this as a given, the differentiation between the design and the engineering's process can be grounded. According to Moonen 2016, the design process is configured to assess variants and to select the most suitable design solutions as a function of the design constraints. The engineering process functions as a method to create a clearer design by giving the design a meaning by assigning rough elements (with properties and attributes) and a higher level of detail (LOD). This clear difference between the function of these two phases implicate the change of character; from more conceptual to a design with a high level of detail.

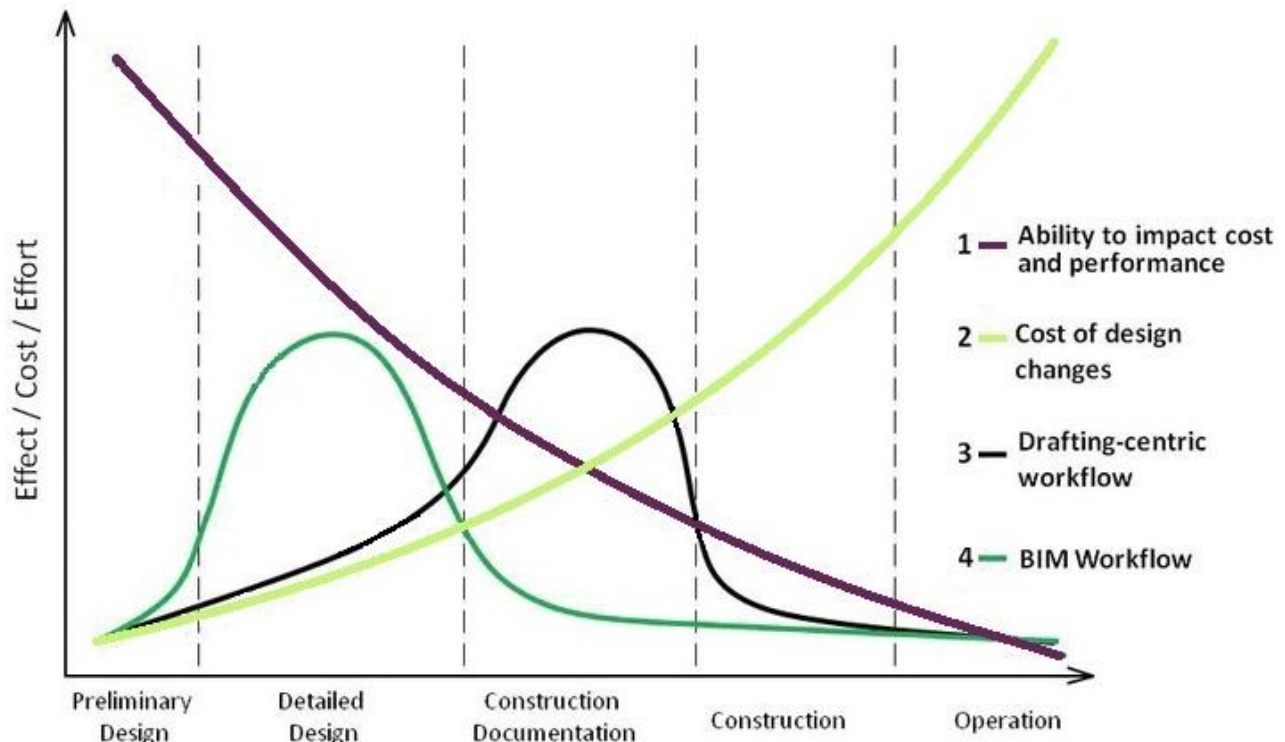


Figure 4: Macleamy Curve (Lu, Fung, Liang & Rowlinson, 2015).

According to Moonen 2016, the difference in the level of detail and level of development run parallel with the elaboration during the design phases. More specific, the level of detail is the definition of how detailed an element is captured within a design of a building model. The level of development is the degree of information and consideration which is put into a geometrical element in a 3d model (BIMForum, 2015). This difference is from great importance given the fact that level of development has a factor of reliability in itself due to the consideration which is put into an element (Moonen, 2016). The amount of variants should be reduced as the level of development will be more sophisticated. This implies that a definition of an element will be more specific and clearer. The level of development varies among the various elements as there are dependencies in importance of development (BIMForum, 2015; Solihin & Eastman, 2015).

3.2.1 Information exchange in the design process

During both the design as the manufacturing stages, a lot of information exchanges will occur. The interaction of information and data is crucial for the quality of a construction project, as there is a clear interaction between information in requirements and design solutions during the various phases of a construction project (Chen & Luo, 2014). The documentation of the interaction between requirements and design solutions is gaining importance as the necessity of proving performance towards clients is growing with the introduction of integrated contracts within the construction industry (Chao Duijvis, 2017; Kim et al., 2015; Pels et al., 2013). The interaction between requirements and design solutions is shown in Figure 5.

It is from great importance to make the right comparison during these interactions, as otherwise the information which is created will not be useful (Moonen, 2016). The management of information, and specifically the exchange of information in a construction project, is an important factor that can influence the quality of a construction project (Eastman, Teicholz, Sacks, & Liston, 2011).

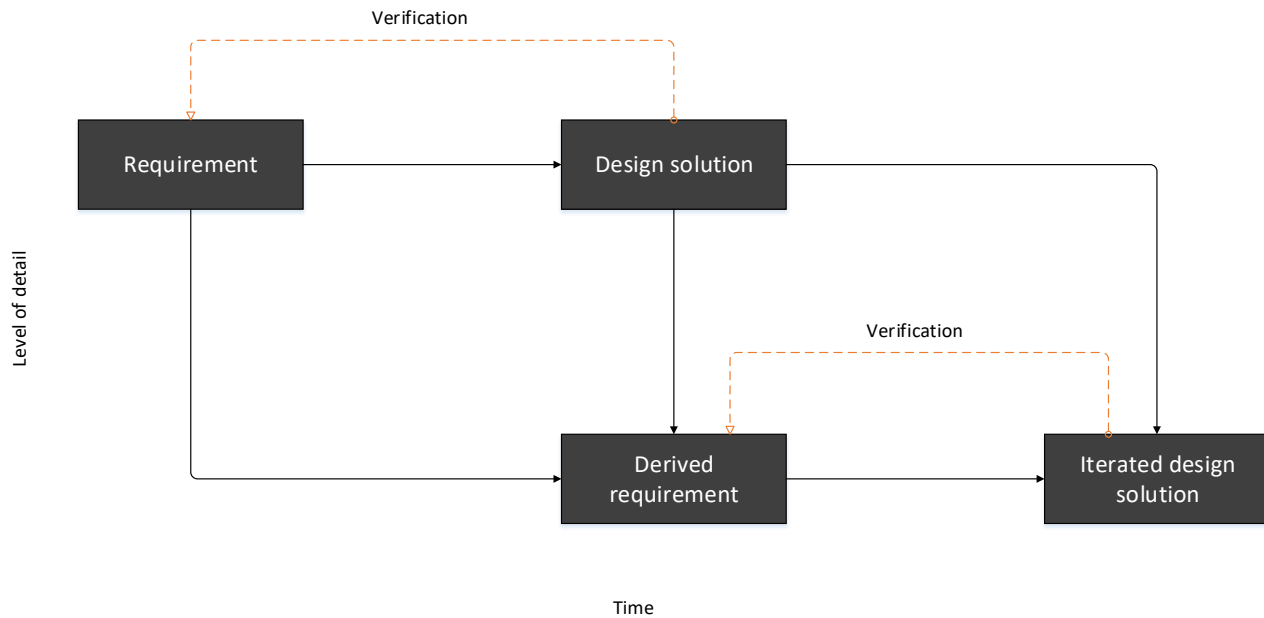


Figure 5: Interaction between requirements and design solutions (Moonen, 2016).

According to Moonen 2016, with the introduction of Building Information Modelling (BIM) the opportunities of managing the information in a construction project have grown. BIM is defined by Eastman as *“a modeling technology and associated set of processes to produce, communicate and analyze Building Models”*. These building models are visualized in three dimensions by means of graphical computer aided design (CAD) models that are structured from objects which contain both graphical and computable information and data. This implicates the pleasant usage of a BIM during the total lifecycle of a system since these models are enriched with the right building information and data. Within a BIM, information can be captured for numerous purposes as a function of a total process. The principle in which the building information is captured and the interoperability of such a model defines how eligible a BIM will be (Moonen, 2016). Still, the way in which the interoperability of information within such a BIM model can be managed has been addressed as one of the major challenges with the use of BIM (Dimyadi & Amor, 2013; Eastman et. al; Yougn Jr., Jones).

The complexity by managing the interoperability of such models is due to the various types of data used in the industry, the unstandardized processes, varying classifications methodologies and the great variety in stakeholders involved (National Institute of Building Science, 2011). The BuildingSmart Alliance has introduced the Industry Foundation Classes (IFC) to improve the

interoperability of the BIM models within the AEC-industry. This IFC standard is an international standard that promotes the possibility of describing buildings throughout their lifecycle by means of neutral file exchange (BuildingSmart, 2013). This creates a big managerial edge whenever various type of project participants are collaborating during design, construction and maintenance processes. The IFC standard reaches the opportunity to improve the collaboration between different domains involved by reducing or discarding the amount of errors that occur during information exchange.

The principle data standard of the IFC is object oriented (Moonen, 2016). The IFC standard is specified by its data schema. The architecture of the data schema is structured by means four conceptual layers. These layers are classified as follows; the domain layer, the interoperability layer, the core layer and the resource layer (Leibich et al., 2013). According to Moonen 2016, the resource layer is the lowest layer where the resource definitions are set. This layer doesn't have unique identifiers as they are defined at a higher level layer. The entities which built up the building are defined in the core layer. More specific, here is where walls or windows (ifcWall, ifcWindow) are defined. The more specialized objects and relationships can be defined by means of the interoperability layer. Here for example, a relationship between a wall and a space can be defined here (ifcRelBoundary). Within the domain layer, the specific concepts towards a domain are defined. For example, information and data regarding construction management are defined here. Here for example, the domain layer can include information and data about values like the cost of an element. A total building model is described in an open data schema by means of these layers.

The applied CAD software which is used by the designing parties involved within a design process can translate their models into an IFC format to ensure interoperability (Moonen, 2016). The building information can then be used for numerous purposes within a construction project. IFC promotes the possibility to use this data among various projects in a same way. This due to the fact that the data can be stored in a standardized way. This opens up the opportunity to use this data for automation of the processes within design phases (Moonen, 2016).

3.2.2 Systems engineering

The implementation of the generic theorem of Systems Engineering is looked upon as this is becoming a more standardized method of working in the construction industry (BNA et al., 2009). This method therefor offers a way to evaluate on the current design process. Systems engineering is introduced within the construction industry to structure and manage the complexity of construction projects. Literature defines various definitions of this generic theorem. The definition of systems engineering, as widely used within the Dutch construction industry, is defined by the International Council on Systems Engineering (INCOSSE). This organ assigns the following definition to Systems Engineering:

“Systems engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the

development cycle, documenting requirements, and the proceeding with design synthesis and system validation while considering the complete problem: operations, cost and schedule, performance, training and support, test, manufacturing and disposal. Systems engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation. Systems engineering consider both the business and the technical needs of all customers with the goal of providing a quality product that meets the users needs” (INCOSE, 2015).

According to Moonen 2016, a few keywords can be identified for proper implementation of systems engineering in, given: Systems thinking, Interdisciplinary, Completeness and Quality. These keywords are essential by implementing systems engineering given their explanation on how the goals of SE can be achieved. The characteristics of systems engineering must be elaborated upon in more depth since this could improve the insight and the understanding of the use of systems engineering within the AEC-industry in practice. Identifying these steps will give sketch a clear overview that communicates on how the information on the requirements interacts with a certain design.

Systems thinking

A system can be described as a holistic whole consisting of interacting parts that work together for a stated purpose (INCOSE, 2007; ISO/IEC/IEEE 15288, 2015). A system is created by people to provide for a certain need within a certainly defined environment (INCOSE, 2015). According to Moonen 2016, the parts of a system can be described by means of objects, people, services or other entities. The parts of a system are mostly defined as objects by implementation of systems engineering within the AEC domain. The used of systems thinking is introduced for a better understanding of the total project or process. Systems thinking is the fundamental basis of systems engineering. The total system is structured by means of layers of subsystems. These subsystems are dissected out of the system, and are used for dealing with complexity in hierarchical matter (Moonen, 2016).

Functional thinking

According to Moonen 2016, the aim of the product development by means of systems engineering is to fulfill a purpose. The functionality of this system can be seen as the fulfillment of the purpose. A system is an answer to a certain group of functions that the product shall accommodate. This is one of the fundamental reasoning why the need of thinking in functionality is important to create a system. Thinking in functions therefor also demands to execute analysis from a larger to a smaller scale which synchs with the top down method of systems engineering (Ministry of Infrastructure and the Environment, 2005).

Client's need central in the process

The need of the client is monitored continuously during the system development. The demand of the client, which represents its need, is the main guidance by the creation of a suitable (product) design. These needs are there for translated in requirements to verify the design. Complying client

specific requirements and design is very critical process (Moonen, 2016). Verification takes place during the whole iterative process by the development of the design to optimize the integration of the client needs in the best way (ProRail, 2015). This approach safeguards the client's needs as a central point within the configuration of the design.

Transparency

According to Moonen 2016, transparency is required within work processes to achieve a higher design quality. Reasoning behind decision making are more likely to be agreed upon if transparency is taking into account during design process. Clear interpretations of reasoning by decision making will be shifted to successive procedures if these are not captured by means of transparency. It leaves both partners and the client in doubts when decision making is not clarified in a rational sense. The justification of design decisions made, whether right or wrong, make processes traceable. Traceability can be very useful by tracing good and bad decision (errors). The open and transparent process are most likely to result in less time loss and a higher quality (Werkgroep Leidraad Systems engineering, 2007).

Decomposition

Systems engineering makes use of a top down approach (IncoSE, 2015). Therefore, decomposition is needed to create an overview of a total system and to get more insight in the complex information and data (BAMinfra, 2008; ProRail, 2015). The eventual tree structure of a system can be created and more insight on a higher level part can be provided by subtracting lower level parts by decomposing the total system (Werkgroep Leidraad Systems engineering, 2007).

Interfaces

There can be assumed that there where different systems or parts of the environment come together, and affect each other through their connection, interfaces can be found (BAMinfra, 2008). The complexity of a project can become clear as the boundaries of different systems can interact by means of interfaces between elements (Moonen, 2016). This interaction can be observed as physical forces, streams and information (ProRail, 2015). The interaction can affect the mutual influence on the total system whenever these interactions are not researched and monitored critically (Moonen, 2016). Manufacturing defaults are often occurring due to this event, this is why monitoring interfaces properly is a critical part of Systems Engineering (Visser, 2011).

Requirements

According to Moonen 2016, the emphasis on requirements management and engineering during the whole lifecycle of a system is essential for the implementation of systems engineering. Only from clear (unambiguous) requirements a solution can be derived which suits all the needs of the client. Unambiguous is the manner in which it is completely clear what is meant by means of one common interpretation (Grant, Kline, & Quiggin, 2009). The exploration of these requirements is there for an essential part of the systems engineering process (Werkgroep Leidraad Systems Engineering, 2007).

Verification & validation.

Moonen 2016 introduces on the topic of Verification & Validations according to the following statement obtained from INCOSE 2007: “The systems engineering handbook as developed by the INCOSE association defines verification and verification as the following two questions”; “Are we building the right thing (validation)?” & “Are we building it right? (verification)” (INCOSE, 2007). The phenomenon of verification and validation are essential parts of the systems engineering process. These processes occur multiple times during the process to regulate the developed elements of the total system. Verification is needed to rest assure that the quality of the created product, whereas validation is required to ensure if the correct product is created. Goals in terms of time, costs and technical specifications can be in danger when the verification and validation processes are not executed by a sophisticated and disciplined approach (ISO/IEC/IEEE 15288, 2015). By this approach the needs that stakeholders require are interpreted better and the process stages are defined more clearly (INCOSE, 2007).

Life cycle approach

According to Moonen 2016, Systems engineering approaches the development of a system with an approach of the total life cycle. Here the total life cycle can be defined as the process from initiation until retirement of the process (Moonen, 2017). A better understanding of the project can be achieved by evaluating the total life cycle of the product (ISO/IEC/IEEE 15288, 2015). In this way the needs stakeholders require are interpreted better and the process stages are defined more clearly (INCOSE, 2007).

Systems engineering process

Numerous field experts, research institutes and scientists have contributed to the theorem of Systems Engineering. The representation of the systems engineering process in its fundamentals has been adopted widely in the research upon systems engineering (Moonen, 2016). The fundamental steps of system creation has been illustrated and can be found in Figure 6. The major elements in this schematic representation can be derived from the interaction between requirements, functions and design elements (Moonen, 2016). The relation between these three elements determine the functionality of the eventual system (US Department of Defense Systems Management College, 2001).

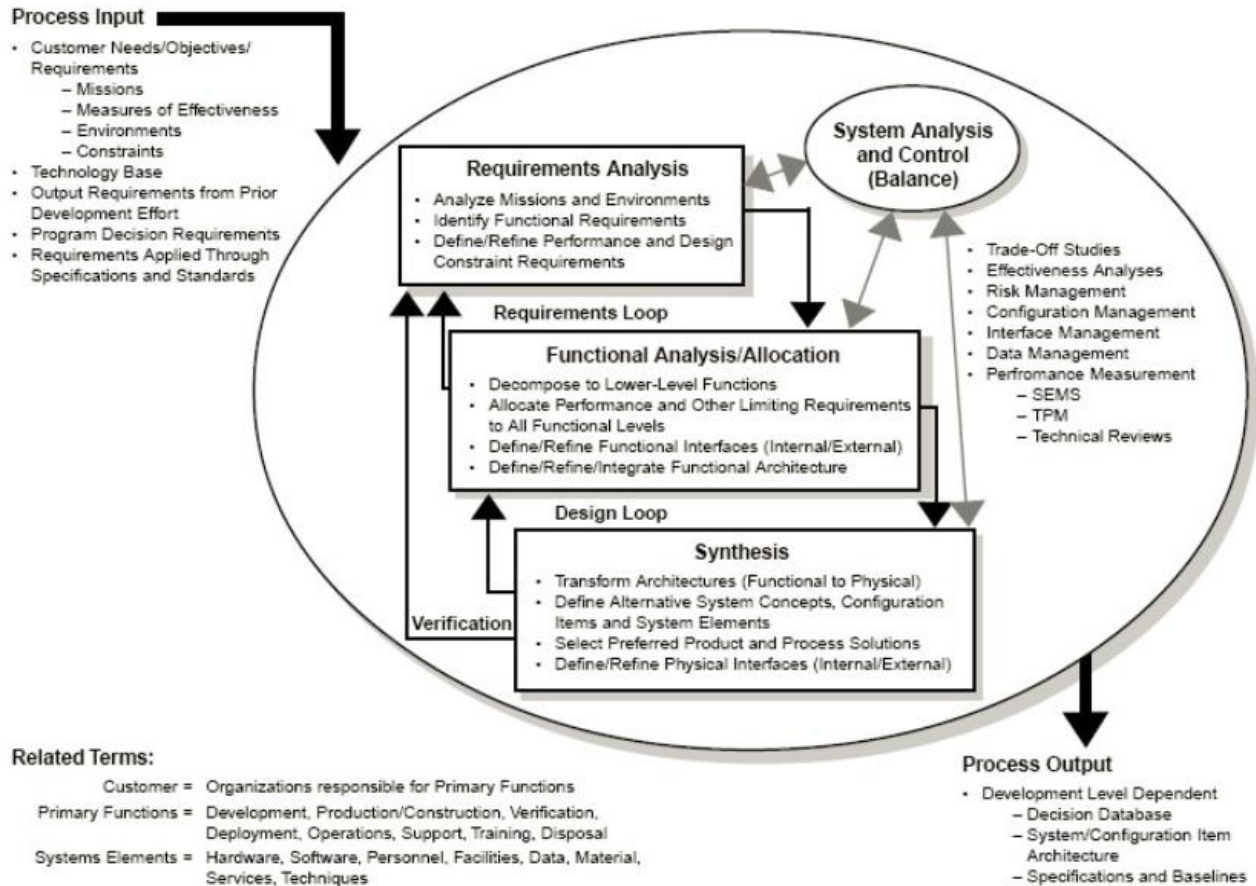


Figure 6: Systems engineering process (US department of defense, 2001).

The V-Model has been used greatly within the systems engineering literature to provide a more detailed representation of the process of systems engineering. The V-model is developed to visualize the top down process within the design loop. Within this V-model, the decomposition of the initial system is realized to give more insight in the total system (Scheithauer, Esep, & Forsberg, 2013). The V-model does not implies that it always visualizes the total life cycle of a project with the use of systems engineering. A more proper overview is achievable by combining representations of the V-model with the total design. This total process can be seen in Figure 7. Within this schematic process representation the phases of the development of a system can be seen as a function of the whole lifecycle. The upcoming sections will elaborate upon the main phases which are relevant for this research. These explanations are based on the work of Moonen 2016.

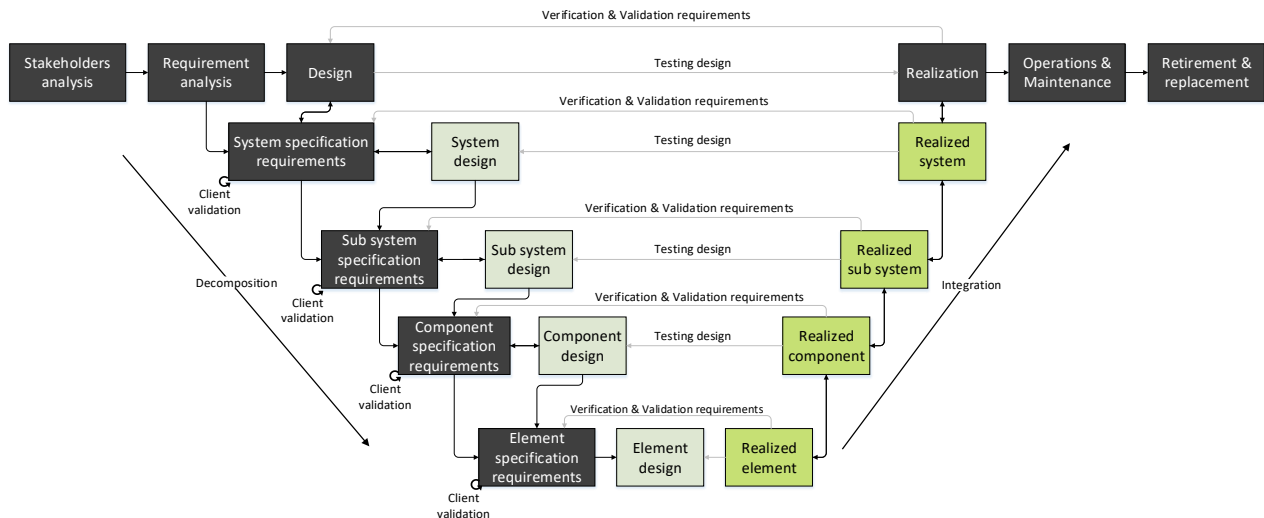


Figure 7: V-Model Systems engineering process extended, based on (Moonen, 2016; BAMinfra, 2008; INCOSE, 2015; Werkgroep Leidraad Systems Engineering, 2007).

Stakeholder analysis

According to Moonen 2016, the key players in a project are identified by means of a stakeholder analysis. This analysis is required to identify how they are affecting the configuration of the system and what their main necessities are in a system. The understanding of the stakeholders' needs is of great importance for identifying what their goals are with a system, therefore proper requirements analysis is crucial (Glinz & Wieringa, 2007). This creates the opportunity to understand, in an unambiguous way, what functionality which stakeholders require for a proper working system. Weak related requirements to stakeholders are a main reason for project failure (Hull, Jackson, & Dick, 2006).

Requirement analysis

According to Moonen 2016, the requirement analysis is defined as one of the most crucial and essential parts of the systems engineering process. This is due to the fact that the understanding of the requirements is defining dictating design constraints (BAMinfra, 2008; Douglass, 2013; ProRail, 2015). Requirements as provided by the client are mostly described in an ambiguous and multi-interpretable manner (Marchant, 2010). The problem related to the ambiguity of requirements can partly be grounded by the fact that requirements are likely to be stated in natural language. This requires extensive linguistic analysis to understand the meaning and the goal that a client had by formulating his need by means of a requirement (Moonen, 2016). These linguistic analyses aim for a better and unambiguous interpretation of the need. A proper validation with the client is needed to ensure that the interpretation is done correctly to reduce uncertainties that mostly reflect in discussions about the interpretation of the need in further stages (Rijkswaterstaat, 2015).

3.2.3 Design phases

The system is realized within the design phases assuming a systems engineering approach. The system is realized with a so called 'Top down' approach. The definition of a top down approach can be described by the fact that the system will be decomposed in to smaller elements. This approach will lead to the configuration of the total system. It is crucial to realize a system that is performing according to the required needs of a client that reflects in the interaction between requirements and design elements of the system (Schaap et al., 2008). This interaction is shown in Figure 8. At all levels of decomposition, an interaction between the requirements and the design is given. Verification of the design complies with the requirements needs and needs to be done at every level of decomposition. This stimulates the possibility to define if the design has the performance as required by the client. A continuation on a mistake can increase an error if verification isn't executed at all levels of decomposition (Rijkswaterstaat, 2015). For this reasoning baselines are defined after all phases within a project. This baseline needs to be verified according to the initial requirements to ensure the quality in a project, and to prevent continuing on biased and defective information and data as a function of the design process.

The definition of a system and building elements is a crucial interaction where information gets related to each other (Moonen, 2017). The definition of the right comparison is essential for a proper working system to ensure the right performance (Moonen, 2017). This interaction between requirements and the eventual performance is visualized in Figure 8.

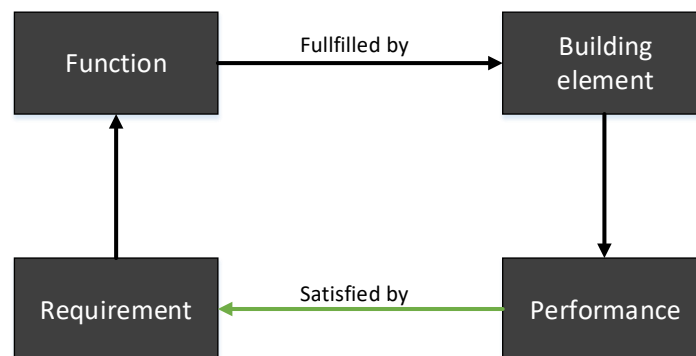


Figure 8: Interaction in design process, based upon Schaap et al., 2008.

Every successive step within the design phase of the systems engineering process ensures the harmonized development of the system and increases therefore the level of detail (US Department of Defense Systems Management College, 2001). This balanced development and increased level of detail during the design phases are visualized by means of the V-Model.

The decomposition of a system reflects in a detailed development and increased detail. Decomposition in the AEC domain is done by means of various methods that mostly rely on the

type, the functionalities and the users of a building. This grounds the reasoning behind the fact that a system can be broken down in various ways in terms of a System Breakdown Structure (SBS). Besides the SBS within a systems engineering process, also other breakdown structures are made according to the same principle. Therefore, the following breakdown structures are used to decompose a project, given: Requirements Breakdown Structure (RBS), System Breakdown Structure (SBS), Work Breakdown Structure (WBS), Organizational Breakdown Structure (OBS) and the Functional Breakdown Structure (BAMinfra, 2008). These breakdown structures are correlated to each other on various levels and define the total scope of a certain project. Within these steps a link towards the system elements are described which should be elaborated on. These objects are linked to a function, certain requirements and to responsible persons.

The design process happens iteratively as explained previously. The use of a certain expert system, in which design decisions as taken in previous projects, could contribute significantly for time management during early design stages. What has been learned from the past could then be consulted by means of an open a smart system that supports reasoning by decision making. When a system is described by a demand specification, variants are made to evaluate which design is satisfying the best solution for the requirements. This is currently done by means of a trade-off matrix to evaluate all the aspects of the variants proposed (Jahan & Edwards 2013). After a decision is made, the reasoning behind a certain choice should be documented and then requirements should be evaluated for a higher level of detail to derive the implications of iteration (Moonen, 2016). This would ensure that applicable requirements are taken into account by evaluating the variants among each other (Moonen, 2016). The steps which are taken in the design process are evolving from conceptual decisions to more detail. This will happen during the subsequent phases of the project. A good example of this procedure is given according to Moonen (2016), where for example the system design of a HVAC concept will be evaluated and a definition will be made about its functionality. In the subsystem design, this system will be iterated into a concept of distribution. In the component design this distribution concept will be drawn in a more specific way and in an element design the products will be selected.

Realization

The manufacturing process can start after the design is defined, verified and validated by the client. This implicates the realization of a system. This realization process is executed on an element level and will result in a bottom up realization of the total system (Moonen, 2016). This realization needs to be tested through verification. Various tests can be applied by verification techniques such as construction test, inspections and measurements. These techniques will ensure that a building functions according to the initial requirements (BAMinfra, 2008). Throughout the realization process, the connectivity of the various elements between the various breakdown structures ensure that the realization is done according to the realization plan. This also ensures that the system is verified according to all relevant and applicable requirements. The WBS is the most important breakdown structure within this phase since the execution of the realization is described within this breakdown structure (Moonen, 2016).

Operation and maintenance

The systems engineering approach can be very useful during the operation and maintenance stage of a system. This due to the fact that the realization of the system is documented. Therefore, a description of the functionalities and the elements is available. This could improve the execution of operation and maintenance processes more easy (BAMinfra, 2008).

3.2.4 Requirements

Requirements as provided by clients function as the input of both process and product design (Moonen, 2016). The goal of defining a requirement is to translate and capture the need of the stakeholders involved to define what functionality the new system must accommodate (Hull et al., 2006). The definition of a requirement by means of the ISO standard of Systems engineering defines that a requirements is a statement that defines a need with associated constraints and conditions (ISO/IEC/IEEE 15288, 2015). A requirement originates from a certain intention and goal, which can be translated into needs (Walraven & de Vries, 2009). If the needs are defined in a clear way, then translations into requirements with a specific product performance can be formulated. This fundamental hierarchy translates the origins of a requirement. This approach should always be taken into account when considering requirements and the performance. This hierarchy is visualized in Figure 9.

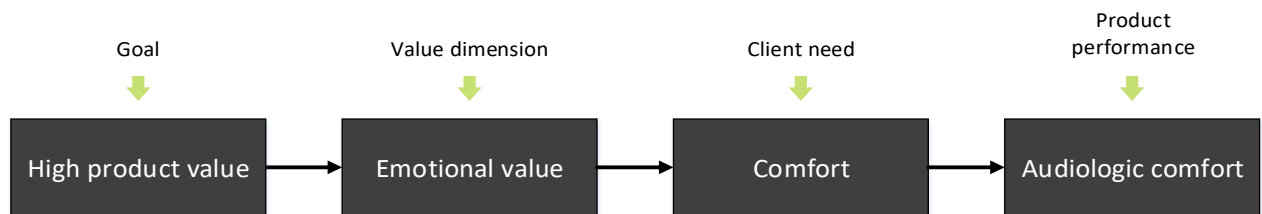


Figure 9: Hierarchy of client needs, based upon (Walraven & de Vries, 2009).

The conditions for a requirement to be used adequately has been researched upon greatly by numerous researchers from various fields of engineering (Moonen, 2016). Fundamentally, a requirement should be unambiguous, measureable, traceable, verifiable and concise (Sparrius, 2014). Various types of requirements have been identified and classified in the research on requirement management and engineering (Moonen, 2016). Three types of requirements can be identified according to Schneider & Berenbach, given: Physical, functional and non-functional requirements (Schneider & Berenbach, 2013). These types of requirements are illustrated in Table 1.

Type of requirement	Explanation	Example
Physical requirements	Describes for examples sizes, property	Windows must have a Rc of at least 3,0
Functional requirements	Describes a desired function	A door must have the possibility to be opened
Non-Functional requirements	Qualitative requirement	A room needs to be cozy

Table 1: Type of requirements based upon (Schneider & Berenbach, 2013).

According to Moonen 2016, the manners in which requirements are characterized in this table show the difference in the interpretation. On one hand, the physical requirement has a clear required value for a property which is verifiable while on the other hand the functional requirements must have certain ability (Moonen, 2016). The last type of requirement is more difficult to see if it complies according to the requirement (Moonen, 2016). For the class of the Non-functional requirements, (ambiguous) fuzzy requirements, it is more difficult to measure whether it complies according to the requirement. To verify these type of requirements in a building model, the requirements need to be suitable for measuring the compliance of the model (Moonen, 2016). This compliancy is defined by the performance of the design relation to the required format. Moonen 2016 states, when requirements are not measurable and therefore not verifiable, problems can emerge as interpretation can play a bigger role due to ambiguity. To make these requirements verifiable, the requirements need to be SMART. The abbreviation SMART stands for: Specific, Measurable, Attainable, Realizable and Time bounded. This means that requirements need to be understandable and prevent ambiguity (Moonen, 2016).

When a requirement is not quantifiable, it is easily affected by the interpretation which can cause major errors during the communication process (Glinz, 2005). Therefore, requirements in the construction sector can also be classified as numerical, relational and qualitative (Moonen, 2016). According to Moonen (2016), numerical requirements are easily reproduced and would cause few problems as the numbers can be made clear. Furthermore, this numerical kind of requirement can be translated into a mathematical equation which can be checked by a computer. This promotes the possibility to automate this process. The second kind of requirement is relational and is a Boolean checking of the requirement (Schneider & Berenbach, 2013). This basically means that whenever the relation is there it is correct and if not, then it is false. The last type of requirement about quality can be arguable which makes it very complex to measure and therefore not quantifiable or possible to check without a lot of interpretation (Moonen, 2016). From these types of requirements a lot of problems can occur due to their ambiguous description and their multi interpretability character. A risk in working with requirements can therefore be found in the interpretation of requirements (Moonen, 2016). Communication with the client and verification of the performance is therefore a crucial part of the whole project (Kiviniemi, 2005).

A good understanding of these requirement is needed given the fact that the meaning of requirements can make a major impact on the design. Requirement analysis is there for an important part of the design process (BAMinfra, 2008). The validation of this interpretation with the client defines if the need of a client is satisfied (Moonen, 2016). Management of requirements can often have little to no attention in a project while iterating the design (Moonen, 2016). As requirements evolve due to iteration and decomposition, the design solution in the end result can shift away from the original goal (Kiviniemi, 2005). According to Kiviniemi, four reasons for the problems related to the management of requirements can be stated. These can be described by the missing connection between requirements and designs, changes in personal during a project,

not directly involved end-users and direct and indirect requirements (Kiviniemi, 2005). Kim et al., 2015 have defined two additional reasons why requirements management and engineering in construction can be difficult. The first reason can be explained by the reasoning and the interpretation behind a requirement that is not documented properly. The second explanation is due to the complexity in requirements that arise from the many types of requirement, spaces and functions which are interrelated with each other (Kim et al., 2015). Malsane et al. have defined the following three characteristics which cause complexity by the interpretation of requirements; subjectivity, inconsistency in terminology and complexity in structuring interrelationships (Malasane et al., 2015). This implies that requirements are prone to the experience of the interpreter, often inconsistent in the terminology and are complex to structure and in the way they relate to other requirements and elements (Moonen, 2016).

According to Moonen (2016), these problems mostly occur by the fact that the requirements are open to one's interpretation. The measurability of a requirement is often stated by means of linguistic descriptions in text. The lack of documentation of the reasoning increases the complexity and the subjectivity of requirements. Another increasing aspect is the fact that the direct relation with the design is often missing. The development of a knowledge based system that describes the relation between the requirements and the design should therefore be very useful to overcome the difficulties existing from the missing relations between the requirements and the applying elements. Also the understanding and the reasoning should be captured in this system to maintain the knowledge which is created during previous projects. A clear overview in the information stream from goal to product performance needs to be synched as this process has many steps of iteration and interpretation. Due to the amount of steps made from need to product performance, this design process remains difficult to manage and to keep close to the desires of a client and the end-user (Moonen, 2016).

Requirement types

Requirements are known to state and manifest various needs. There has been found that a variety of requirements exists. To be more specific on requirement types and their properties, there has been chosen to introduce an overview of the variety of requirements that exist within construction projects. To define a valid list of the requirements types, the structure of developing a requirement must be followed. As previously mentioned within this report, a requirement is created to define a certain need of a certain client. A requirement can therefore be assumed to be a translation of a need that corresponds to a certain value on its own. This is a very crucial, somehow forgotten principle. From a certain need, for example the need of a 'cozy building' a translation (attempt) to a requirement will be made.

Initially, requirements are known to be applicable for spaces. Spaces are a non-tangible objects, this space requirement needs to be satisfied by the elements which can define a certain (cozy) space. We can assume that we can express a (cozy) space by certain objects that function as the space its borders. The interaction between a certain space and certain objects; and together with that the iteration from a space requirement towards an object requirement is essential for defining

a performance (Moonen, 2016). This total composition of interactions between space requirements, spaces, object requirements are visualized within Figure 10.

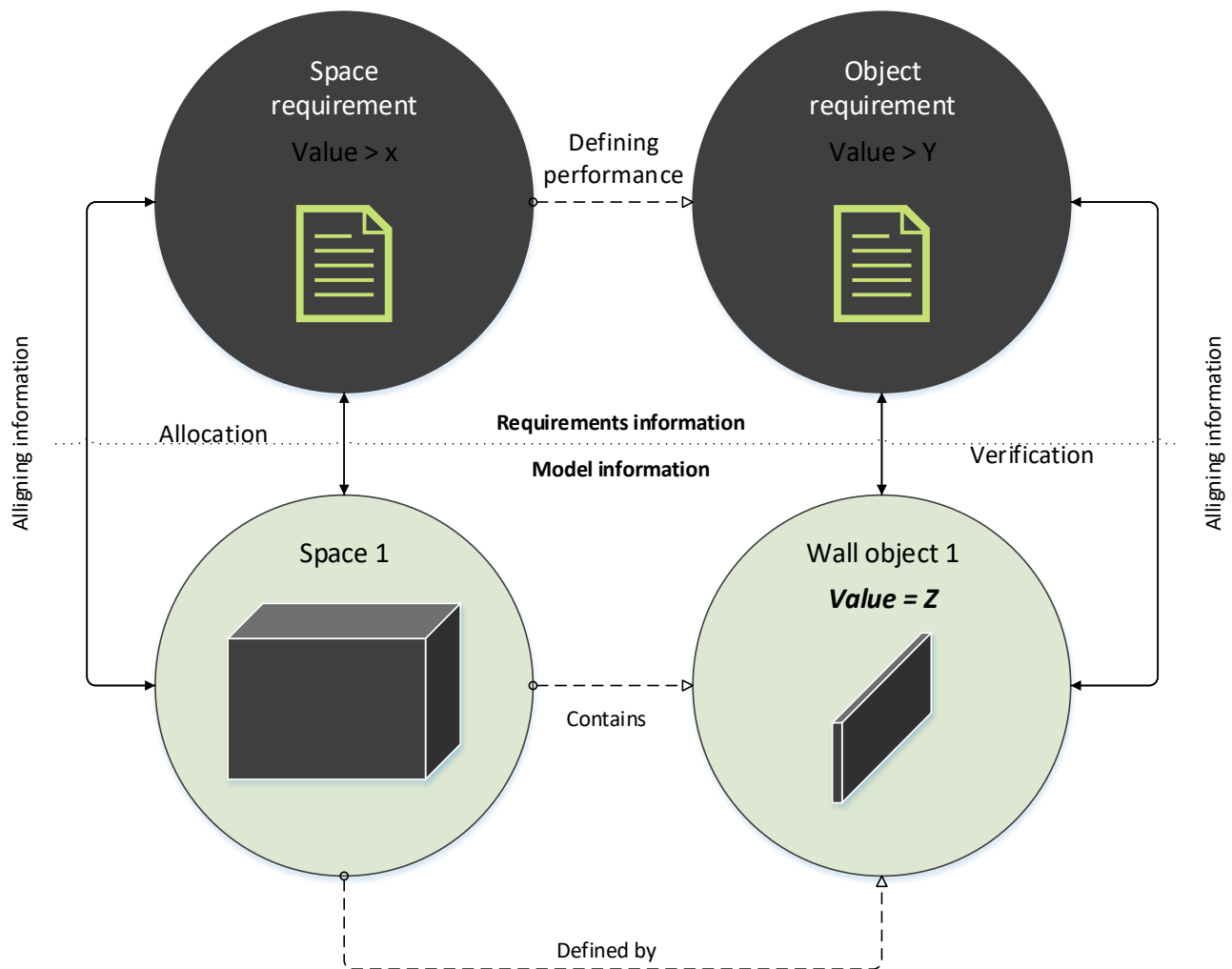


Figure 10: Interaction between information in requirements and objects (Moonen, 2016).

For the creation of a complete list of requirements types, existing projects need to be investigated. This is a tough job, especially when decisions in regards requirement management and engineering have never been logged and captured. Here, the findings as stated within the work of Moonen (2016), have been used for the sake of brevity. Moonen (2016), has investigated five existing projects upon the various client requirements. The requirements within these projects are managed with the use of a relational database (Relatics). In these environments, the requirements of the clients have been translated into manageable interface. In Table 2, a description is given about the projects which are used for the definition of the requirement types.

	Project 1	Project 2	Project 3	Project 4	Project 5
Type of building	Government building	Government building	Office building	Education building	Government building
Client type	Government	Government	Developer	Educational institute	Government
Size	13000m2	6700m2	32000m2	40000m2	80000m2
Total requirements	4000	1533	3580	1812	13830

Table 2: Description of analyzed projects (Moonen, 2016).

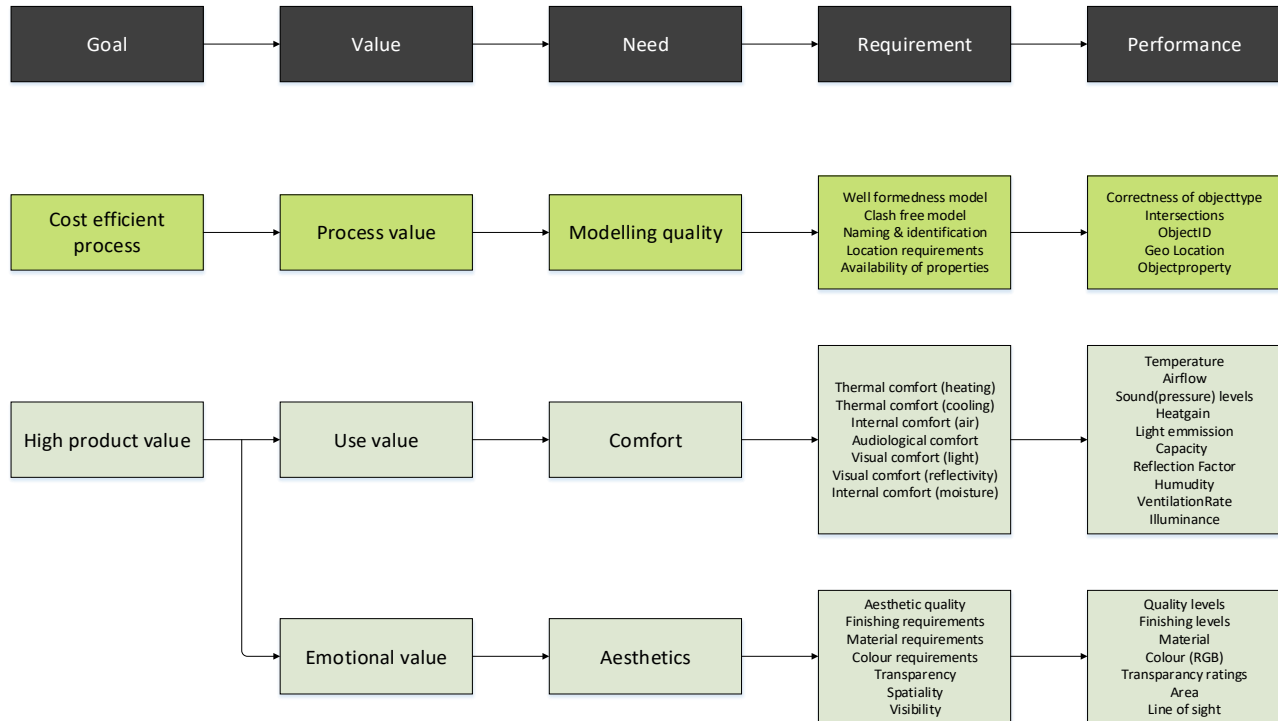


Figure 11: Requirement type classification (Moonen, 2016).

Various requirements are coming forwards given this data set. The aim of a requirement is always to define a certain need that a client desires. A requirement starts from a certain function and value which is required to be present in the building. These define certain needs like for example a ‘cozy building’. An analysis on the variety requirements of requirements, executed by Moonen 2016, can be seen Table 3.

	Project 1	Project 2	Project 3	Project 4	Project 5	Average
Project Year	2016	2015	2015	2014	2014	2014,8
Total amount of Requirements	3982	1533	3530	4742	12984	5354,2
Square meters	13.000	6.700	32.000	40.000	80.000	34,34
Requirements per m2	0.306	0.229	0.110	0.119	0.162	0,19
Value requirements	9.8%	10.4%	11.6%	12.5%	6.2%	10.10%
Object requirements	22,00%	19.9%	41.80%	37.8%	14.5%	27,20%
Space requirements	57.7%	31.4%	23.6%	64.1%	58.3%	46.6%
SMART requirements	17.7%	73.8%	x	8.8%	x	33.4%
Requirements usable in BIM	17.3%	13.7%	7.2%	15.1%	7.9%	12.2%

Table 3: Data analysis outcome (Moonen, 2016).

3.2.5 Verification

Both literature as the AEC-industry assume several definitions in regards to verification. The definition of verification, according to the ISO/IEC/IEEE 15288, is stated as follows; “Verification is a confirmation through the provision of objective evidence that specified requirements have been fulfilled” (ISO/IEC/IEEE 15288, 2015). This translates into the question for the design process; “is the design correct?”. The verification process can be identified as a feedback loop to complete the design process (Moonen, 2016). The essence of a generic verification process is visualized in Figure 12. This representation visualizes the fundamental procedures within the verification process. The definition of a verification process should there for be done adequately as otherwise an evaluation will not have any value (Marchant, 2010). Validation of a requirements should be done to enrich the validation process with value. This approach ensures that the verification can be executed correctly. The validity of a requirement remains a difficult endeavor (Moonen, 2017).

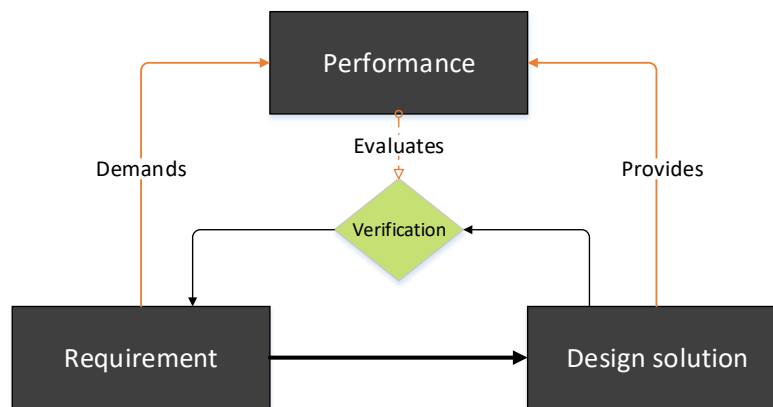


Figure 12: Essence of the verification process (Moonen, 2016).

Shishko & Aster have defined procedures to ensure that requirements are unambiguous, traceable, correct and well defined. Whenever his definition is adequately defined and given before a design is made, then the design will improve and the verification process will much become easier (Shishko & Aster, 2007). Haskins has defined the steps to undertake to ensure proper verification to ensure the completeness of the verification process. This process consist of three steps; Preparing, Performing and manage the result of verification (INCOSE, 2015). These steps will be elaborated upon within de following paragraph.

The first step is structured by means of a definition of the strategy and corresponding tactics for verification in a project in relation to costs and risk. Within this step, the definition of what should be verified (requirements, characteristics etc.) is firstly defined. After this is done, the procedures will be assigned which ground with what they will be verified. The constraints that flow from this procedure will then be defined towards the execution. Lastly, during the preparation of the verification, the availability of information should be taken care of to ensure that the execution can be done smoothly. This whole procedure should be documented within a verification plan. In this verification plan the definition of verification, the success criteria, the used verification method,

the required information and data and the enablers is captured (INCOSE, 2015). Secondly, the verification should be executed according to the plan and the results should be analyzed. These results should be communicated. This communication process functions as an evaluation on which actions should be taken to cope with non-complying elements.

If we take a closer look to the root of failures and defaults within the verification process, the following reasons can be found according to Marchant, 2010; inaccurate or defective requirements, ambiguous, incorrect allocation of requirements and even missing elements. These reasoning cause errors if the verification process isn't adequately evaluated and validated. These reasons can result in a positive outcome of verification, but are actually failing due to the incompleteness or incorrectness (Moonen, 2016). This presence can be inaccurate and can lead to extensive rework and costs if discovered in later stages of the project. Underestimating this procedure can therefore provoke higher project variance. This phenomenon is quite often the case within the AEC-industry. The traditional method of working within the AEC-industry heavily relies on the workmanship of constructors to deliver a product (building) that is suitable for usage (Moonen, 2016). Verification of requirements has become an important process within the design process of building projects due to the increasing complexity and the introduction of integrated contracts (Bouwens Nederland, 2014). Underestimating the importance of verification within a design process can result in major mistakes. A big opportunity lies here to improve the design process to ensure that designs are complying with the requirements and improve the quality (Moonen, 2017).

3.2.6 Conclusions

This part of the literature review has been conducted to allocate the research problem and objective as a function of the overall design process. The traditional design process has been set against the systems engineering process. The systems engineering approach is more sophisticated, if executed correctly, relative to the traditional design processes. This sophistication flows from the structured approach towards the specification and traceability of linguistic chunks of text, as obtained from client's brief, into requirements that reflect in structured design decisions. This makes design decisions traceable and justifiable as a function of both product design and during operation and maintenance. This could possibly imply why clients are stating the SE approach as one of their process requirements, especially in case of complex demands where market parties are somehow unfamiliar with.

It seems to be crucial to express the design task in terms of a problem context, the project itself, and the desired system. These three aspects should relate to the content of the client's brief. These aspects could contribute to the definition of boundary conditions and assumptions that dictate how the requirements can be formulated to achieve a certain system performance. However, the SE approach is relatively new within the AEC-industry. This could make it hard to convert the design decisions as taken from the past into a set of standardized knowledge that can be used for decision making. It is crucial to have a set of boundary conditions and assumptions before translating ambiguous client's wishes within a system by complex design tasks. It is essential in these cases to

find dependencies in words within the sentences of the client's wishes towards elements of the system prior the SE process.

There has been observed that the effectiveness of knowledge systems could be positioned after the client's briefing stage, and prior to the systems engineering process. Consulting a system in which knowledge has been captured, in terms of boundary conditions and assumptions in regards to the translation of client specific requirements in product specifications, could contribute to manifest the client's wishes in product configurations. The use of a certain knowledge system could possibly contribute to a head start for the participation of tender. This is the moment where such systems can prove their functionality as a support tool for specifying requirements prior to tenders to gain a head start within a competitive environment. Programming a system according to experience and knowledge from the past, by means of a knowledge system, can be very useful in processes where a little actual information is available.

3.3 Knowledge Management

The AEC industry is induced to work more effective and efficient due to the complex demands that arise from the client's need within the architecture, engineering and construction domain. Traditional managerial approaches lack in their synchronicity to the current demand from the market. A potential solution to counteract these possible processual deficiencies is by the introduction of Knowledge Management. This approach can be implemented on an organizational level within firms to optimize their governing influences. The AEC industry has introduced numerous (static) techniques in the past, based on traditional KM techniques, but these systems require a lot of maintenance. These systems are also lacking in their capabilities to actively share knowledge due to their static nature. With the fast development of research on the domain of Building Information Domain, new opportunities raised to create knowledge management systems. These systems can be consulted during decision making procedures as a function of their application within the variety of both design and manufacturing processes. The development of Building Information Modelling (BIM) contributed a lot to lucrative application of KM within the AEC domain. This due to the fact that a BIM is basically an informational database in which design decisions have been stored as a function of a certain demand specification. This development might reach the opportunity to extract crucial information from previous projects, or shared projects on the World Wide Web, that can be used by decision making according the a knowledge system. A BIM can provide a very specific and unique source of information and data as it generates, manages and captures the data created during the life cycle of a building. This principle reaches the possibilities to gather data by data mining approaches to promote KM.

The changes from an industrial driven society to a knowledge based driven society has let the aspect of knowledge to be the foremost important resource of a company. Therefore, sharing knowledge within a certain company has become more important than ever (Johannessen, Olaisen & Olsen, 2001). This due to the assumption that a lot of administrative benefits might be achievable on an organizational level by the right interpretation of the available information and data. The reuse of existing organizational knowledge attained by previous experiences can reduce a lot of time spent on problem solving, and can therefor increase the quality of work which results in a competitive advantage on the long run (Rekveld, 2017). Therefore, managing the knowledge that is spread around the organization is from great importance. The management of knowledge is especially important for the companies within the AEC-industry. This due to the high amount of engineering tasks, as these are highly knowledge and experience driven (Deveraja, 2015). Companies should be able to leverage their knowledge in order to maintain their sustainable competitive advantage over the competition to make their business more profitable.

The fundamental definition of Knowledge management (KM) is difficult to articulate and to quantify because due to the fact that it withholds elements of disciplines of both *hard* and *soft sciences* (Abaljaber et al., 1998). It seems that there is no consensus on what KM is (Rekveld, 2017). Research initiated by MIT (Abaljaber et al., 1988) reveal that different articles are defining different solutions in terms of KM. The same research, a small alteration of the definitions of Frappaolao and Tomes (1997) is proposed, given: *"KM is a tool set for the automation of deductive or inherent*

relationships between information objects, corporate users and business processes". Logically speaking, it is from great importance there is a common understanding of what knowledge is and how this can be obtained. The upcoming sections will fundamentally demarcate where knowledge originates from, how that it can be defined and which types of knowledge exist.

BIM data as knowledge source

Building Information Modelling is a very promising development within the architecture, engineering and construction (AEC) industries where numerous researchers are currently investigating upon. With BIM technology, it becomes possible to generate a digital and accurate virtual 3 dimensional model of a building with enriched information and data. Whenever BIM is implemented correctly within the whole construction process, then the computer-generated model contains precise geometry and relevant data needed to support the construction, fabrication and procurement activities needed to realize the building (Eastman et al., 2011). The realization phase is not the only phase where BIM technologies can be useful. BIM technologies can also be very beneficial during the operation and maintenance phase of the building (Davtalab & Delgado, 2014). A useful definition of BIM was described by Campbell (2006). He defines a BIM as an intelligent simulation of architecture that exhibits the following six key characteristics, given: (1) Digital; (2) Spatial (3D); (3) Measurable (quantifiable, dimension-able, and query-able); (4) Comprehensive (encapsulating and communicating design intent, building performance, constructability, and include sequential and financial aspects of means and methods); (5) Accessible (to the entire AEC/ owner team through an interoperable and intuitive interface); (6) Durable (usable through all phases of a facility's life).

A BIM model manifests itself by building components that are enriched with information and data that describe how they behave and are consistent and non-redundant data (Rekveld, 2017). Building components are modeled as objects that have a digital representations and data about what they are. These can be related with computable graphics, data attributes, parametric rules and descriptions on how they behave. This makes it possible to create analyses of the building and its usage in work processes. This BIM model also contains coordinated data. Besides the BIM model, another very important part of BIM is the interoperability between parties of a certain project team. This safeguards the fact that every team member is assured to have access to the latest project data. It makes it possible to allow every member to have access to all the data. A cloud based server, such as a BIMserver, is the most used technique to ensure that project data is both shared in real time as it is accessible from different locations.

According to Rekveld (2017), it is possible to allow every member to have access to all the data, these data need to be: real time data exchange and share in a predefined format. There are two primary approaches for the predefined format, given: (1) use a proprietary file format and therefore stay within one software vendor's product; or use another product that is allowed by the vendor, or (2) use different vendors that can exchange data using nonproprietary file format that is a universal supported standard. The advantage of the predefined format is that it allows for tighter integration among products in multiple directions. For example, a change in one model

results in a change in all other linked models. The recognizable disadvantage is that every team member of the project team is forced to use the programs of the specific vendor. This could potentially affect the investments a lot considering both licensing and training of the members. The second approach would solve the disadvantage of the first approach but the disadvantage of this approach is that the current universal standard, Industry foundation Classes (IFC), is not designed to carry all relevant data.

Still, the implementation of Building Information Models (BIM) has proved to be in use by the enhancement of the performance of AEC projects (Rekveld, 2017). Rekveld (2017) states that the BIM is a *“shared knowledge resource for information about a facility forming a reliable basis for decisions during its life-cycle; defined as existing from earliest conception to demolition”*. It has contributed to the improvement by the communication of the design between various stakeholders, by enabling the identification of clashes ahead of time, by enabling the simulation of the construction sequence, and by the improvement of the communication between various craft subcontractors and the general contractor (Deshpande, Azhar & Amireddy, 2014). According to Rekveld (2017), building information models are inherently parametric, data-rich, object based representations of the facility being designed and constructed. Thereby, building information models can be both conceptualized as centralized, interconnected data stores which can contain design and fundamental construction information about the various disciplines involved within a certain construction project. Rekveld (2017) further states that this centralized and integrated nature of the design information can potentially provide a very context rich platform for the capture, storage and dissemination of the knowledge generated during the design and construction processes.

One of the principle requirements of an effective knowledge management system is its ability in communicating and capturing knowledge effectively across various phases of a construction project (Dave and Koskela, 2009). BIM models are uniquely qualified as a knowledge source due to the fact that BIM models can be used over the whole span of the construction project and even evolve and are able to capture the knowledge as soon as the knowledge is created (Deshpande, Azhar & Amireddy, 2014). Although BIM models are qualified as great knowledge sources, the knowledge within these models is not explicit. Therefore BIM models can be seen as sources of embedded knowledge (Rekveld, 2017). Embedded knowledge is knowledge that is locked in processes, products or artefacts according to Argote & Ingram (2000). Even though embedded knowledge can have an explicit form, such as BIM models, the knowledge itself is not explicit, the implications of the embedded knowledge are not immediately clear. (Gable & Blackwell, 2001). Rekveld elaborates upon this fact by stating that the knowledge itself has to be made explicit and usable to be able to use the embedded knowledge as a source for knowledge management. To promote this, big data techniques will be introduced. The definition of big data and the associated techniques will be explained in the upcoming sections.

3.3.1 From data to knowledge

A good understanding of the concept of knowledge and knowledge taxonomies is important due to the fact that theoretical developments in the field of KM are affected by the distinction among the different types of knowledge (Alavi & Leidner, 2011). The fundamental concepts of data, information and knowledge are closely related (Kock et al. 1997), and it is commonly known that knowledge has a higher level than information, and information has a higher level than data (Tuomi 1999). According to Rekveld (2017), data can be defined as symbols that represent the properties and attributes of objects and events without any added interpretation or analysis. Data simply exist and has no significance beyond its own existence and they can exist in any form, usable or not. They do not have meaning of their selves (Ackoff, 1989; Ackoff, 1999). According to Davenport and Prusak (2000), *“data are a set of discrete, objective facts about events”, and “provide no judgment or interpretation and no sustainable basis of action”*. Data are syntactic entities and patterns without meaning, and exist in usable or non-usable forms without significance beyond their own existence (Aamodt and Nygard, 1995; Bellinger et al. 2004). Uriarte (2008) states, *“data have no meaningful relation to anything else, since they are missing a context”*.

On the other hand, information is assumed to be data that have been given meaning by means of relational connection. This enrichment can be very useful; but it is not mandatory to be so. Information can be explained as structured data with meanings, which is generated from the interpretation process of data (Aamond, Nygård., 1995). Ackoff (1990) defined information as *“data that are processed to be useful, providing answers to ‘who’, ‘what’, ‘where’, and ‘when’ questions”*. The last definition to be given is that on knowledge. Here, knowledge can be assumed to be refined information (Rekveld, 2017). Rekveld (2017) states that knowledge is the appropriate collection of information, in such way that its intention is to be useful. Knowledge is a deterministic process according to Rekveld (2017). When someone is memorizing information, then they have amassed knowledge. This knowledge has a certain useful meaning to them, but it does not provide for, in and of itself, an integration such as when it would infer further knowledge (Ackoff, 1990; Aamond and Nygård, 1995). Data are a carrier and storage of information and knowledge along with a media for information exchange and knowledge transfer (Kock et al. 1997). Kock et al. (1997) states that information is descriptive and related to the past and the present, while knowledge can be used to predict the future within a certain range. The role of knowledge is to facilitate the processes of transforming data into information through data interpretation, deriving new information from existing through elaboration, and acquiring new knowledge through learning (Aamodt and Nygård 1995).

Tacit and explicit knowledge

The knowledge as captured within organizations can be identified by means of two dimensions, given: tacit and explicit (Nonaka, 1994). Nonaka (1994) states that tacit knowledge is rooted in action, experience, and involvement in a specific context. Here, the cognitive element is referring to an individual's mental model consisting of mental maps, beliefs, paradigms and viewpoints. According to Rekveld 2017, the technical component consists of concrete know-how, crafts and skills that apply to a specific context. Pozzali & Viale 2015 state that tacit knowledge consists of

professional expertise, individual, insight, experience, and creative solutions. Junnarkar and Brown (1998) propose that *“tacit knowledge is that which is implied but not actually documented”*. More specific, knowledge can be tacit not because one is unable to articulate it; but because it has not been captured yet. This perspective is very useful according to Rekveld (2017) because it suggests that some tacit knowledge may be more valuable when made explicit than other. The goal of knowledge management would not be to explicate all tacit knowledge, but rather to assess first the existing tacit knowledge and determine that which has the most value before trying to make it explicit (Rekveld, 2017).

The class of explicit knowledge is articulated, codified and communicated in symbolic form and/or natural language. Rekveld (2017) states that most explicit knowledge exists in forms of technical or academic documents, such as manuals, mathematical expressions, copyright and patents. This “know-what” or systematic knowledge is readily communicated and shared through printed documents, electronic methods and other formal ways. On the other hand, explicit knowledge is technical and requires a level of academic knowledge or understanding that is gained through formal education. Explicit knowledge is codified, stored in a hierarchy of databases and is accessed with high quality, reliable, fast information retrieval systems. Whenever codified, explicit knowledge assets can be reused to solve many similar types of problems or connect people with valuable, reusable knowledge (Smith, 2001). A reason for companies not to invest in KM is due to the fact that sharing processes often require major monetary investments in the infrastructure needed to support and fund information technology (Hansen et al., 1999).

Knowledge conversion and creation

Aside from the tacit-explicit distinction of knowledge another distinction between dimensions of knowledge was identified by Nonaka (1994) (Rekveld 2015). The dimensions individual and collective (or social) knowledge, in combination with the tacit-explicit dimension, can be used to distinguish different kinds of knowledge conversion and creation (Rekveld 2017). Nonaka (1991) dimensioned four types of knowledge conversion on the SECI (Socialization, Externalization, Combination and Internalization) model. These four fundamental types are socialization, externalization, combination and internalization. The first type, ‘Socialization’, is converting individual tacit knowledge to group tacit knowledge. The type ‘externalization’ tacit knowledge is made explicit. The type ‘combination’ is conversing separate explicit knowledge to systematic explicit knowledge whereby ‘internalization’ is the conversion from explicit knowledge to tacit knowledge.

	To tactic knowledge	To explicit knowledge
From tacit knowledge	Socialization	Externalization
From explicit knowledge	Internalization	Combination

Table 4: SECI model of knowledge conversion (Rekveld, 2017).

Goal of knowledge management

Within the previous sections, knowledge and knowledge management have been discussed. There has been defined what knowledge and knowledge management are and from what importance it is. Still, the actual goal of knowledge management is, has not been covered yet. The fundamental goal of knowledge management (KM) is to connect knowledge providers and knowledge seekers to provide value creation and create sustainable competitive advantage (Abaljaber et al., 1998; Alavi & Leidner, 2001). According to Rekveld (2017), Sustainable competitive advantages can be achieved through resources that are valuable, rare and imperfectly imitable. Here, the resources can be property-based or knowledge based. Property based resources are legally controlled by a specific firm where knowledge based assets are protected because they are often subtle or difficult to understand or copied by outside observers.

In a study executed by Davenport, De Long and Beers (1997) four business objectives that fulfil the goal of KM are identified, namely: (1) To capture knowledge; (2) To improve knowledge access; (3) To enhance the knowledge environment; (4) To manage knowledge as an asset.

Capturing knowledge can be done by the creation of KM repositories. These archives consist of structured documents with knowledge embedded within them, stored in a way that they may be easily retrieved by queries. According to Rekveld (2017), much better access to knowledge can be facilitated by improving the processes of knowledge transfer between individuals and between organizations. Transfer and use of an enhanced knowledge environment can be achieved by proactively facilitating and rewarding knowledge creation. Knowledge should also be managed as an asset. This can be achieved in various ways. On one hand, some companies could include their intellectual capital in the balance sheet, while on the other hand other companies leverage their knowledge assets to generate new income or reduce costs. Knowledge can be part of a certain business means by means of various application within a certain firm.

3.3.2 Conclusion

This part of the literature review has been conducted to measure how knowledge management can counteract to the translation and specification of ambiguous client specific requirements. Experiences from the past seem to be of great importance to translate data into information, and information into knowledge. However, the difference between tacit and explicit knowledge is crucial within this procedure. The field experts within the AEC-industry are withholding a lot of tacit knowledge. Their knowledge is captured within their minds and are not made explicit. Explicit knowledge implies fundamentally that the knowledge is captured by means of certain techniques that is accessible by human. Explicit knowledge could bridge the unknown to the known. Generic theorems are known to be captured in literature and other sources. However, the specific knowledge of specialists within the AEC-industry are often not captured by means of standards and semantics. This makes it hard to use such knowledge. This implies partly the tradition of intuitive decision making. Clients are often unaware how decisions are made previous and during the design stages. Whenever justifications for decisions are asked, experts tend to reconstruct their procedure

rather than deliver the exact procedures. However, there must be mentioned that the introduction of BIM and particular tools such as relational databases and common work environments are gradually contributing to solve this problem area.

The rise of electronic relational databases seems to contribute to the storage of data and information on projects as a function of time. This promotes the traceability and justification of procedures due to the translation and specification of client specific requirements, especially when a systems engineering approach is introduced to structure these procedures. However, the content of these databases is often static. The content can be consulted for manual queries, but are often not enriched with the right knowledge for decision making. The formal notations within these databases are insufficient. This implies the linguistic chunks of text which are not easily transformable into decision variables for decision making. The approaches, such as systems engineering, that are used within relational databases to design these environments are fundamentally structured. However, the information where this approach consists of does not contain semantics. Semantics as in standards within its data. This might be required to find laws during data analysis which is required to achieve semantics. Assuming a sentence to be a token, then we would find a variety of formats on how these are formulated sentences are structured currently. There is no structured way in which linguistic descriptions are formalized, which make it hard to use this information for automation purposes.

The unstructured content of such relational databases might imply the urged need for a knowledge system in which design decisions are captured electronically by means of a formal notation. This explicit knowledge can then be consulted for support by a variety of processes within the design process. This especially for the translation and specification of client specific requirements previous to the early design stages. Such knowledge systems can be introduced prior to the systems engineering process. The implementation of such systems could promote user client interaction, especially under a scarcity of information where clients are known to be unprofessional. Here, unprofessional implies the unfamiliarity of the client to specify his needs as specific as possible. It is of great importance during these iterative processes to support clients given the fact that they are unfamiliar with the business and design processes within the AEC-industry. Misinterpretations due to ambiguous client specific requirements can reflect on wrong design decisions. Harmonizing the client's needs and the expectations according to boundary conditions and assumptions seems to be crucial during these stages. Consulting knowledge, therefore, by means of certain automated knowledge systems, can be of great importance during these stages to provide trust, exploit business processes, and safeguard product performances.

3.4 Natural language constraints

3.4.1 Constraints within engineering

According to Niemeijer (2011), the majority of constraints in the building industry are specified using a natural language, such as Dutch or English. Examples of these include building codes and requirements that are included in the client's brief. There are many rules that must be obeyed by the design for a certain consumer product (Halman et al. 2008). According to Niemeijer 2011, on one hand some of these rules are derived from human morphology; a phone must be small enough to fit in your hand. However, on the other hand, some of these rules will be marketing-based; a maximum cost requirement. Yet another source wherefrom design rules originate are laws and regulations; the safety requirements on cars (Niemeijer, 2011). All these rules are expressed by means of constraints that the final design must satisfy in order to link the demand to expectations. These constraints can be formulated by means of ranges in which design decisions need to be taken. Constraints dictate certain bandwidths in which design decisions needs to be taken in order to satisfy the client's needs. Currently, checking whether all of these rules have been satisfied is, in most cases, executed manually (Niemeijer, 2011). This is very labour-intensive given the large amount and variety of existing rules. Developing a way to automate this checking procedure would immensely benefit this process. However, this could possibly imply that it is required that building regulations need to be formalized in an objective manner so that they can be verified by a computer. According to Niemeijer (2011), a large subclass of all building regulations can be formalized, but there are still some crucial exceptions. Regulations such as "the architectural quality of the addition must correspond to that of the surrounding buildings" have no objective interpretation, as "architectural quality" is an ill-defined term: does this concept refer to technical quality or the aesthetics? The Dutch institute, Concepten Bibliotheek Nederland, has contributed since early 2011 to define a lot of these concepts (CB-NL, 2014). Still, there is no single accepted concept and thus the rule cannot be formalized. The CB-NL is striving for glory but still a lot of concepts are missing which make this ontology not usable in reality. There needs to be mentioned that this will evolve as a function of time. However, the computer will be able to check a sizeable amount, if not the majority, of the regulations, removing the need for people to worry about the trivially checked rules and giving them more time to focus on questions of aesthetics (Niemeijer, 2011). Therefor we assumed that design rules that can be formalized will be referred to as constraints (de Vries et al. 2000). The word constraint has many different definitions in the numerous fields of engineering. In this thesis however, the definition given in "Constraint specification in architecture" (Niemeijer, 2011) is used: *"a CSP [Constraint Satisfaction Problem] is a problem composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values the variables can simultaneously take"*. Constraint satisfaction is the process of arriving at a design solution that satisfies all of the constraints (Dohmen 1995).

Designing products under a set of constraints, with the purpose to optimize the product performance, is a challenge by engineering in general. The intention to program products by means of automation under a set of constraints is addressed numerous research. According to Niemeijer

(2011), many industries are using several methods and techniques to automate design verification. In electrical engineering, for instance, many steps of the design process are partly or fully automated, including placement, routing and power optimization. In the field of software engineering there are several ways of applying constraints to a unit of code, among which static typing, unit testing and code contracts. They mostly have the same goal, but use a different methodologies and techniques. All three mentioned approaches can be seen as a way to handle constraints. They formalize the criteria that the code should satisfy and can be checked automatically; thus preventing the programmer from making certain types of mistakes. In a lot of respects, mechanical engineering is similar to building design. In both disciplines, three-dimensional objects are designed that have to obey a series of constraints. Despite the similarities, there are also clear differences between the two. Mechanical engineering has a much stronger tradition of storing design semantically rather than only as the resulting geometry. There are several, often complementary, avenues of research in this field such as parameterized solid modelling, feature based modelling, component-based or modular design, and constraint-based design (Niemeijer, 2011). These will not be discussed in depth due to the demarcation of this research.

Constraints within AEC-industry

Given the fact that construction projects within the AEC industry are getting more complicated, due to their technical complexity as a result of the high pre-defined set of requirements, the D&E are more apt to develop strategies which are profitable to integrate these aims as effective and efficient within the requested product. These requirements are basically the results of the demand that the ordering parties have which can be formulated as the boundaries of restrictions in which they desire their product to be developed in. These boundaries of restrictions form the ranges of possibilities in which the D&E can move in order to program and design the corresponding product.

To manage design requirements, therefore, the following conditions must be met: (1) monitoring to ensure that a design solution satisfies the requirements and (2) updating of the requirements when project information that affects those requirements changes (Kim, Kim, Cha, & Fisher, 2015). Within this research project, we assume that the product requirements evolve in a set constraints that need to be satisfied. According to Niemeijer (2011), the building industry, and more specifically, the architecture domain, has seen little adoption of constraints, at least not in the sense that they are (fully) automatically checked in comparison to other industries. Naturally, building designs have to comply with a multitude of constraints, such as building codes and functional and technical requirements that follow from a client's brief, but verifying these is still a manual process in most cases.

Only in the past 30 years have constraints started to get some traction. A few example projects in which constraints are used are: 'Digital Dormer' (Leeuwen, Jessurun, & de Wit, 2004) whereas legal constraints are used for the design and permit approval of dormers, and the 'SMARTcodes' (Wix, Nistbet, & Liebtich, 2008) that checks if the building models are in harmony with building codes (Niemeijer, 2011). Recently prototypical model view checkers, for model instance validation of Industry Foundation Classes (IFC) models (Zhang, Beetz, & Weise, 2015), have been developed by

use of constraints. The architecture, engineering and construction industry are exposed to several types of constraints. According to Niemeijer (2011), architectural constraints can be subdivided into many different types based on the topic of the constraint. Some of the more common types are (Niemeijer, 2011):

- *Geometrical constraints*: Constraints on dimensions; e.g. the width of a certain door
- *Structural constraints*: Constraints regarding the strength of elements; e.g. loadbearing capacity of a material
- *Building physics constraints*: Constraints about the climate of a building; e.g. the required humidity in a room
- *Material science constraints*: Constraints in regard to the properties of materials; e.g. porosity of a material
- *Financial constraints*: Constraints on the cost of parts of a design or the design as a whole; e.g. the budget
- *Aesthetic constraints*: Constraints intended to achieve a certain look; e.g. the corresponding amenity

According to Niemeijer (2011), three types of constraints can be identified. The first type are the quantitative constraints (e.g. the height of the wall must be less than 3 m). The second type are the qualitative constraints (e.g. windows cannot overlap). The third type, hybrid constraints, combines elements of both. In this research the (fuzzy) hybrid constraints are the main focus for study.

3.4.2 Methods of using constraints

There are several ways to interpret constraints. According to Niemeijer 2011 there are two ways of dealing with constraints, depending on who creates the design; the user or the computer. Both approaches result in a design that satisfies the constraints, still they have different properties and attributes and have different application domains. The first way to use constraints is by means of *constraint solving*. The brief description of an example by use of this method is taking the constraint as an input and trying to find a design that satisfies them (Kelleners 1999; Eggink et al. 2001; Belbidia, Alby, 2003; Bohme, Cárdenas, 2006; Donath and Bohme 2007). The second way to practice constraints is to produce a certain design and check afterwards whether the design meets all the constraints, and adjust the design there where necessary (Niemeijer, 2011). This particular method is called *constraint checking*. An automated constraint checking system will only be able to check constraints that can be computed. This requires that constraints are both *decidable* and *computable* (Davis, 1985; Sipser 1996). Decidability means that the function can be evaluated in finite time. The constraints within the AEC-industry typically fall in one of two categories in terms of computability: they either are simple guidelines or rules of thumb that can be quickly calculated; or they require a computationally intensive numerical simulation (Niemeijer, 2011).

3.4.3 Constraint entry

Within the AEC domain, designers are the ones who will be entering the majority of constraints on a day-to-day basis. This group can also be classified by means of gradients to express the division

of experience and knowledge within this group. However, given this assumption, the Graphical User Interface (GUI) should be designed with this particular group their requirements in mind. The goal is therefore to find a method of constraint entry that is easy for this group to work with. Myers et al. 2006 states the following alternatives on this query which will further within the upcoming sections.

Synthetic language-based constraint entry

The first possibility is to use a certain formal language in the form of a programming language. This can be seen as a natural choice according to Niemeijer (2011) because the amount of expressive power required of the constraint system is similar to that of a (simple) programming language and because programming languages are commonly used to express rules in many different domains. The main advantage of this option is that the implementation is relatively easy. In addition he states, it is likely that at least a majority of all constraints can be formalized using a programming language, based on the use of programming languages to encode constraints in other industries. Niemeijer (2011) also states that the main disadvantage of this option is that programming languages are very formal and require a great attention to detail in order to correctly express oneself, which a lot of designers will likely not be used to. Besides from the precision required, there is the additional issue that many programming languages have a syntax that will not be familiar to non-programmers. Niemeijer (2011) elaborates upon this statements by the following example: in Java the translation of the constraint “The height of windows in brick walls must be between 1 and 2 m” might result in the following code:

```
if (wall.material == materials.Brick) {  
    for(window : wall.windows)  
        assert(window.height >= 1 &&  
            window.height <= 2); }
```

This piece of sample code reveals a few examples of syntax that differs from natural languages, such as curly braces to define scope and the use of && instead of and (Niemeijer, 2011). Some of these issues could be solved by using an Application Programming Interface (API) or a Domain-Specific Language (DSL) targeted at defining architectural constraints (Spinellis, 1999). This could possibly reduce the amount of unfamiliar syntax the designer has to deal with. Given the previous example of the sample code, the constraint then might be expressed as something along the lines of:

window.height between 1 and 2 for window in windows of wall if wall made of brick.

A decent example of a DSL that focusses on the reduction of unfamiliar syntax, to a point of representing it like it is written in English is the Inform 7 programming language (Niemeijer, 2011). It is a programming language specifically designed for creating textual adventure games. A short extract of some sample code (Short 2011):

“The Law Library is north of the Great Dining Hall. “Many [books of precedent] line these walls, containing every kind of contract that can be made to bind every kind of soul. A hole in the floor descends to the other, less savory portion of this place.” Some books are scenery in the Law Library. Understand “shelves” and “books” and “contracts” as the books. The description is “It is not as though you would understand the language in which they are written.” The great contract book is a thing in the Law Library. Understand “contracts” as the contract book”.

This code sample defines a room into two objects that are positioned in that room, and gives those objects descriptions and synonyms, so that for example the command “look at shelves” will produce the description of the books rather than providing an error message that the meaning of the word shelf are unknown (Niemeijer, 2011).

Natural language-based constraint entry

Natural Language Processing (NLP) takes the concept of removing unfamiliar syntax to a new level. This due to the fact that it allows the designer to enter the constraints in a natural language, such as English. This is very different, from a technical standpoint, in comparison to programming languages and DSLs. NLP discards the requirement for training on the part of the designer, since he or she can use the language where he or she is familiar with. However, it increases the difficulty of the implementation significantly, as natural languages are far harder to interpret by machines than programming languages. This problem occurs since natural languages have not been designed with automation by interpretation in mind. Using the previously defined constraint again, we could express it in any of the following, and a multitude of other, ways (Niemeijer, 2011):

- The height of windows in brick walls must be between 1 and 2 m
- Windows in walls made of brick must be between 1 and 2 m high
- The height of any window in brick wall must be higher than or equal to 1 m and lower than or equal to 2 m

According to Niemeijer (2011), the first and foremost difficulty in interpreting natural language is the presence of ambiguity. The exact meaning of words can depend on the context, unlike programming languages. Thereby, there are different types of ambiguity (Hutchins 1992), given:

Category ambiguity

This prompts by ambiguity regarding the grammatical category (noun, verb, etc.) of a word. This can be grounded by, for instance, the use of the word set in the following sentences: “I set the glass on the table”, “They are part of a certain set”, “Is she set?”

Homography

This type of ambiguity can be described by two words that contain the same spelling which are having a different meaning. Interpret, for instance, the following sentences: “Her ear was infected” and “She ate an ear of corn.”

Transfer ambiguity

This case of ambiguity goes for the same word that are having different meanings in different languages. Compare for instance “I had a chat with her” and “le chat est sur la table.”

Structural ambiguity

This type of ambiguity can be described by one sentence having multiple different interpretations. Given, for example, the following sentence: “Flying helicopters can be dangerous” can mean both “It can be dangerous to fly helicopters” and “Helicopters which are flying can be dangerous.”

According to Niemeijer (2011), ambiguities can be resolved by means of different methods and techniques, such as context and real-world knowledge. However, these remain hard to simulate. Supporting natural language input can be made way more feasible by restricting certain language constructs, such as metaphors. Niemeijer states the following general rule; *“The more formal and specific the language used, the easier it is for a computer to interpret”*.

Visual constraint entry

The three categories (programming language, DSL, natural language) as mentioned according to Niemeijers work “Constraint specification in Architecture” within the previous paragraphs cover different types of text-based constraint entry. However, this is not the only possible method since it is also possible to use a graphical interface (Niemeijer, 2011). One way of doing this by means of this technique is to represent the constraints as trees, mirroring their internal structure (Myers 1990). Examples of this technique include ConMan (Haeberli 1988), Microsoft’s Visual Programming Language (a programming language for a virtual robotics environment) (Microsoft 2011) and Yahoo Pipes (Yahoo! 2011), which is a manner to customize RSS feeds.

This particular approach has the advantage over text-based constraint entry (Niemeijer, 2011). This due to the fact that the capability of such system is exposed to the user, given the fact that all the blocks that are available for use are listed in front of him or her. It is way harder to predict whether a certain expression will be supported or not by use of a text-based system (Niemeijer, 2011). The downside of this technique is the readability, especially with more complex trees available the function of the constraint will not be immediately that natural (Niemeijer, 2011). Another method that can be used to solve this issue is a hybrid between the tree structure and natural language solutions (Niemeijer, 2011). According to Niemeijer, here, the principle is to construct natural-language sentences from blocks. An approach similar to this is used in Lego Mindstorms NXT (National Instruments 2011), an environment for programming Lego robots.

An alteration on this method is to use a 3D visual programming language, such as presented in “The CUBE language” (Najork and Kaplan 1991). However, the practical use of this seems very limited, as it is not easy to quickly see the meaning of a rule (Niemeijer, 2011). Thereby, it complicates interaction with the constraint since a 3D environment requires orbiting as well as panning and occlusion may prevent the entire constraint from being visible at once (Niemeijer, 2011).

3.4.4 Conclusion

This part of the review of literature has been conducted to allocate the research problem and development objective(s) as a function of automation. Client specific requirements are known to be formalized and answered whenever their corresponding specifications satisfy the specific need. This implies that specifications are constraining the decision bandwidth of the D&E. The challenge prior to the early design stages of requirement engineering is mostly related to constraint solving rather than constraint checking. During these stages, solutions are required to solve, and therefore satisfy the client's need. In later stages, during verification and validation of requirements for the variety of object levels, requirements are rather checked than solved. This is a very crucial difference in principles that will be accommodated within the prototypical system its functionality.

Requirements can constraint several aspects, depending on its domain of application. Within the AEC-domain, however, constraints are most likely to be categorized in linguistics, legal, geometrical, structural, building physical, material technical, financial, and aesthetical aspects. These aspects can all contain numerical and non-numerical specifications within their properties and attributes. The process to distill a specification for a linguistic description seems to be very error prone, especially in cases where D&E are unfamiliar with the type of linguistic descriptions. This due to the fact that requirements are not always stated by means of numerical expressions, but rather as linguistic constraints where numerical specifications need to be derived from. Having a system in which former translation procedures have been stored in, what can be consulted for queries, might be a very useful technique to reduce errors during these processes. This could reduce or even discard the categorical, homography, transfer and structural ambiguity by interpreting such requirements that contain linguistic constraints. Natural language is not designed with automation in mind. This contributes to the 3rd layer of ambiguity. This 3rd layer of ambiguity occurs by feeding the computers with natural language constraints.

The variety of techniques to process natural language are often very complex from nature and labor intensive. This is especially the case for domain related language where few or insufficient libraries are developed for. Machine learning can therefore be a challenging job, given the fact that certain libraries need to be built up from scratch. There were libraries exist, formats, standards, and semantics often need to be revised and synchronized. This can be laborious and therefore expensive in practice. For the development objectives of this research initiative, there has been chosen to use 'hash-tables' rather than 'natural language processing' as the technique to process natural language due to the sake of brevity and experimental nature of this development attempt. In this attempt, words, definitions, classes, and specifications are tokenized. This is the fundamental hierarchical data structure where the sentences (client specific requirements) will be dissected with. This is also closely related to the formal notation that will be developed later on to fill, enrich and store data and information within the prototypical system its database as a function of knowledge gathering.

4 In-house practices

4.1 Motivation

The goal of this research initiative is to investigate the design process, the current practice of requirement interpretation and translations into product specifications, and the possibilities to introduce automation for the translation of non-functional client requirements into product specifications. The fundamental goal of this initiative is to improve this specific process prior and during the early design process. To achieve this, besides the review of literature, a review of current practices in the design process is required. This approach could possibly identify automation as a pragmatic mean within the design process. Therefore, review on the current in house practices is initiated and executed to create a practical environment. This review on in-house practices is initiated and executed in collaboration with several field experts. Several interviews are held with field experts to demarcate an accurate representation of the current translation procedures. This is done to link and confirm the knowledge of the design and verification process in practice. This approach brings this research closer to the origin where problems are occurring, and where scenarios can be defined to introduce automation. The interviews are short from nature, and are structured according the demarcation of this research initiative.

4.2 Interview

Within this chapter, the research questions will be included as part of this qualitative research. The obtained knowledge from the interviews are used to address the actual situation, along with the problems that the experts are coping with during the translation of non-functional requirements into product specifications in early design stages. This research focuses on elements from the client's briefing stage and the design stage(s) where non-functional requirements are being interpreted and translated into product specifications. The goal for the interview per research question, and sub questions, will now be discussed.

- 1) What client specification procedures are there in use within the design process, and how does Systems Engineering support these procedures?
- 2) What variety of client requirement types are known within the design process, and which of these carry risk in terms of non-conformity?
- 3) What is the current practice in the AEC industry for translating client specific requirements into product specification, and how do verification procedures safeguard these?
- 4) What can automation, for translating client requirements into product specifications, contribute to the design process?
- 5) What are the current techniques within the AEC-domain, by means of automation, to translate product requirements into product specifications?
- 6) Is it possible to develop a method that translates and stores physical, functional, and non-functional requirements into product specifications by means of automation?

1) What client specification procedures are there in use within the design process, and how does Systems Engineering support these procedures?

There are two parts which will be reviewed upon, given: the design process where the non-functional requirements will be translated and systems engineering within the AEC-industry. The current procedures in regards to the integration of client specific requirements will be analyzed as a function of the design process. The defaults and failures, which most often occur during the design process and Systems engineering process, are discussed upon in the interviews to measure what types of issue are actually occurring. The required adjustments for process improvement are also treated. The fundamental preconditions and system requirements are also treated for (evolutionary) prototyping. The goals are:

- Identification of the problems and their origins that occur during the interpretation and translation of requirements into product specifications in the early design phase;
- Identification of the problems that are occurring prior and during the implementation of systems engineering prior and during the design process;

2) What variety of client requirement types are known within the design process, and which of these carry risk in terms of non-conformity?

The goals related to this research question are to discover and capture what kind of requirements are known, how these can be categorized, and how a requirement is structured. To improve the process of working with requirements, the problems with verification of requirements are also treated. The goals are:

- Explore how client specific requirements are structured;
- Explore, categorize and capture the different kinds of client specific requirements in construction projects;
- Define which type of requirements is provoking the most problems of interpretation and translation procedures into product specifications;

3) What is the current practice in the AEC industry for translating client specific requirements into product specification, and how do verification procedures safeguard these?

The total process of requirement interpretation, translation and verification needs to be outlined in order to analyze where the use of automation can come in to practice. These overall processes are therefore required to be evaluated upon. Here, the essentials for a good verification process will be discussed to identify the conditions of good verification within a design process. The relation

with the design process is carefully investigated. The requirements of which the verifications that are known to be the hardest and most risk full will also be treated. The goals are:

- Identification of the types of errors that are occurring, and where they originate from;
- Identification of key elements for a proper verification;
- Relate the design process to the verification process;

4) What can automation, for translating client requirements into product specifications, contribute to the design process?

The implications for automation within interpretation and the translation during early design processes needs to be observed. This would make it possible to set a scope, by means of system requirements, for prototyping. This development process will be treated within the next part of the research. These preconditions are discussed together with the pros and cons of automating these procedures. The goals are:

- Identification of the improvements required within the design process prior the introduction of automation;
- Definition of the pros and cons of automated requirement translation into product specifications;
- Definition of the preconditions for the automation of translation procedures;

5) What are the current techniques within the AEC-domain, by means of automation, to translate product requirements into product specifications?

6) Is it possible to develop a method that translates and stores physical, functional, and non-functional requirements into product specifications by means of automation to develop a certain (semi) automated system?

Here, a brief exploration towards the use of information systems in relation to natural language processing and constraint specification is executed. The experiences from the past along with the actual approach by processing natural language and constraints, as obtained from demand specifications, are addressed and treated within these questions. The goals are:

- Identification of the actual methods and techniques to interpret, convert, and capture information and data as obtained from the client's brief;
- Identification of the trials and problems of these initiatives to explore the "*The quickest wins*" for system development.

4.3 Definition of subjects

The goals for the previous interview questions can be classified to 3 subthemes. These subthemes are the design process, knowledge management, and natural language constraints in Architecture, Engineering and construction. The goal per subject will now be treated:

Design process and Knowledge Management

The design process and knowledge management will firstly be analyzed. Treatment of this subject could provide fundamental insight in the current practice by translating client requirements into product specifications. This could determine where the biggest problems are originating from. Procedures in relation to communication, file and information exchanges are closely observed. How data, information and knowledge is processes and captured, by means of the variety of techniques, is also questioned upon. In this way a clear insight can be given in what sense mistakes occur. This gives also the opportunity to measure the synchronicity between the domains of Building Information Management and Systems Engineering in regards to the design process.

Natural language constraints within the AEC-industry

This subject positions requirement translation procedures in relation to constraints specification. This could identify the variety of methods and techniques that are used in practice for translating non-functional requirements into design constraints. The current practice and use of information systems, expert systems, and knowledge based system will be discussed. From here on, the fundamental system requirements in relation to software development can be treated.

4.4 Interview results

The interviews as held, functioned as a research instrument additionally to the literature review. The literature review was introduced to measure how science is covering this research problem, and the interviews were held to measure the same problem(s) in practice. Field experts with different backgrounds have been interviewed. The interviewees work at certain departments of Systems Engineering & BIM, and fulfil the roles as principle systems engineer; systems engineering program manager; and verification and validation manager.

The goal of this interview was to measure how the design process can be improved by looking into the process of design and verification to observe how the information flows are treated within this process. The observations obtained from this process could address and position the use of a (semi)automated system as a function of the translation procedures. The combination of the types of interviewees broadened the scope and created the possibilities to address the research scope. The interviews were held in a semi structures way. Initially, there were guidelines of questions that were followed. However, the interaction during these interviews provided opportunities to deviate from these questions to obtain more specific information and examples. Therefore, sub questions were created during the interviews in regards to the main interview questions.

The interviews have been recorded and transcribed for analysis per research question, this made it possible to draw conclusions per answer of a respondent. These conclusions have been captured

and have been reflected between the answers of the different respondents. The conclusions per question have been evaluated upon. Finally, the conclusions per question are merged into a total conclusion per subject. This chapter ends by sharing the observed findings by means of answers on the initial research questions.

4.4.1 Design process

The introduction and use of integrated contracts implicates evidence of product performance. Proving the product performance is of great importance. The information which provides the specific type of proof is required to be valid and consistent. The respondents pointed out that the main problem in the design process can be found during the information streams during client specification procedures. There are a few information streams which are identified. The first category is the *customer requirements information*; that is crucial. This is found as the stream where the translation is made from customer requirements to preconditions for design. Creating the information on design solutions implies answering the customer requirements. The right interpretation and understanding of the client specific requirements are essential in order to achieve the product performance and functioning as the client desires. The parts where requirements are applying to need to be clarified. This approach on regulating information streams, provides the opportunity to structure information in order to allocate this to a part of a design. The definition on which parts of the product specific requirements apply dictates where the answer should be given.

Allocation of the requirements is required in order to link the specific need to a specific part of the design. Here, a problem arises given the urged need for a system design before allocation can be executed. It is very difficult to come up with a proper system design in early design stages, especially by a lack of design competence. For doing this, rough bandwidths in which designs can be configured need to be formulated. The allocation should be done with precision to harmonize the relation between client specific requirements and system configuration; this is a crucial process. Major problems can occur from missing allocations to parts of the design when this allocation procedure is not done adequately. There is found that a system is already configured within this stage form constraints as derived from product requirements. In this (roughly) configured system, the relations between elements is already made from information which is derived from the requirement specification. There can be stated that whenever a clear definition of the system is missing, mistakes can occur during the design process which makes it even harder to structure the proof process of the product performances.

However, there are conditions which need to be met before the total process can be proven to be working. Firstly, verification due to the correct interpretation of the requirement needs to be achieved; this is very crucial. Requirements need to be defined and captured unambiguously in accordance with the client. Then, the second step is the allocation of requirements. The system needs to contain the right information on its right places, it needs to communicate the same information as captured in the first (conditional) step. Therefore, a good interpretation and

allocation are the first measures to prevent this. There is also a variety of other information that needs to be allocated to objects and requirements. The fundamental reason why this process needs to be executed securely is to enrich both the model and the information, by harmonizing them with the same information. The alignment in regards to the allocation of requirements and objects, acting disciplines, level of risk, level of detail, responsibilities, applicable design phase(s), and verification procedures need to be treated coherently.

4.4.2 Interpretation of requirements

Field experts found that it is common sense that clients often don't know what they exactly want, how it should look like and how their product should function. This especially during the early design stages. Here, the experiences of designers and engineers should come in to practice in order to capture the specific demand specification as clear as possible. The content of client's briefs are often ambiguous and hard to process. This due to the fact that chunks of natural language are used to describe functions and performances of the desired product. The interpretation of a design can vary greatly among interpreters as the requirements are often written in natural language. This process is very critical since both interpretation and client desires need to be synched and captured; this is mainly the bread and butter within the domain of architectural design. This procedure is especially complex by demand for products where both designers and contractors are unfamiliar with. The validation from the client of the interpretation of a design is therefore of great importance. The possible reason why deviations occur is the ambiguity in the definition of specific requirements as stated in the client's brief. In order to prevent design mistakes, the interpretation needs to be discussed and captured before formalization. This is very crucial, since this could contribute to minimize and prevent contradictions in later phases which are most likely very unprofitable.

In cases where the demand specification is assumed to be complex, especially when both the designing party as contractor aren't familiar with the type of product that is asked, extended requirement analysis should be introduced. The main findings why this extended requirements analysis procedure isn't done within the AEC industry can be explained by the following fundamental reasons. The first reason is that the AEC sector is used to start designing straight away and adjusts its design during the iterative design process. This causes the insufficient time that there is taken to correctly, fully, interpret the need of the client. The second reason can be explained by the fact that the investment costs of extended requirement analysis are earned back after a tender is won. Not every tender is won though, this makes it an unprofitable procedure to introduce for each project. Therefore, it is very important to have specialists reviewing the necessity of such extended requirement analysis since this can be very profitable for contracting complex projects, especially by collaborating with specific clients. Understanding the client needs is a core element to win a tender since this dictates the fundamental design constraints in which design decisions need to be configured. Numerous researchers are contributing to science by investigating on strategies to exploit this opportunity. The plurality of applicability of a certain

requirement is also known to create difficulties by interpretation of requirements. Numerous requirements as stated within the clients brief could be filled in by multiple elements. A requirement can be filled in by a combination of different objects; a requirement can have interfaces with multiple objects. Thereby, an object can have multiple requirements applied. Designing the complying object is very complex if these two things aren't distinguished clear enough.

The interviewees are asked to distribute requirements in classes. They were simply asked to distribute the requirements according to simple, hard, or complex by interpretation. It has been found that the interpretations and the plurality in applicability of non-functional requirements are known as the essential factors which affect the complexity of a requirement. This can also be found in requirements which are identified as the most complex requirements; the non-functional requirements. The non-functional requirements are designated as the most complex requirements. This due to the fact that they need stimulation to prove their performances, especially by multiple interfaces. More specific, there has been found by the answers of the respondents, that comfort and aesthetic related requirements are the most failure sensitive by programming.

The following schemas visualize the schematic representation of the requirement interpretation and translation procedures. The experts are known to implement these approaches as means to interpret, translate, allocate and verify requirements as a function of product design. The upcoming schemas are practical representation of these workflows.

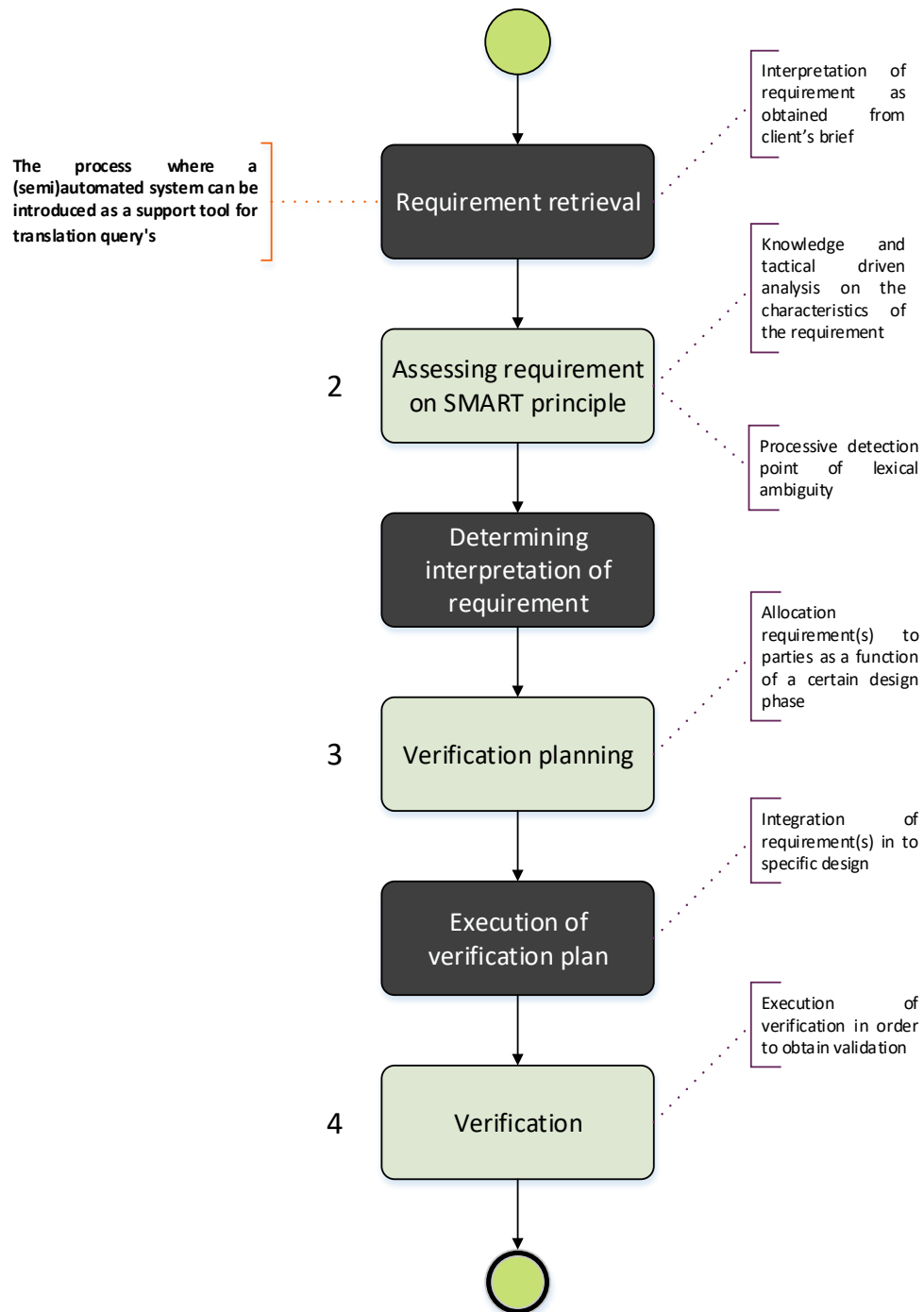


Figure 13: Activity diagram 1: The interpretation, translation and verification process.

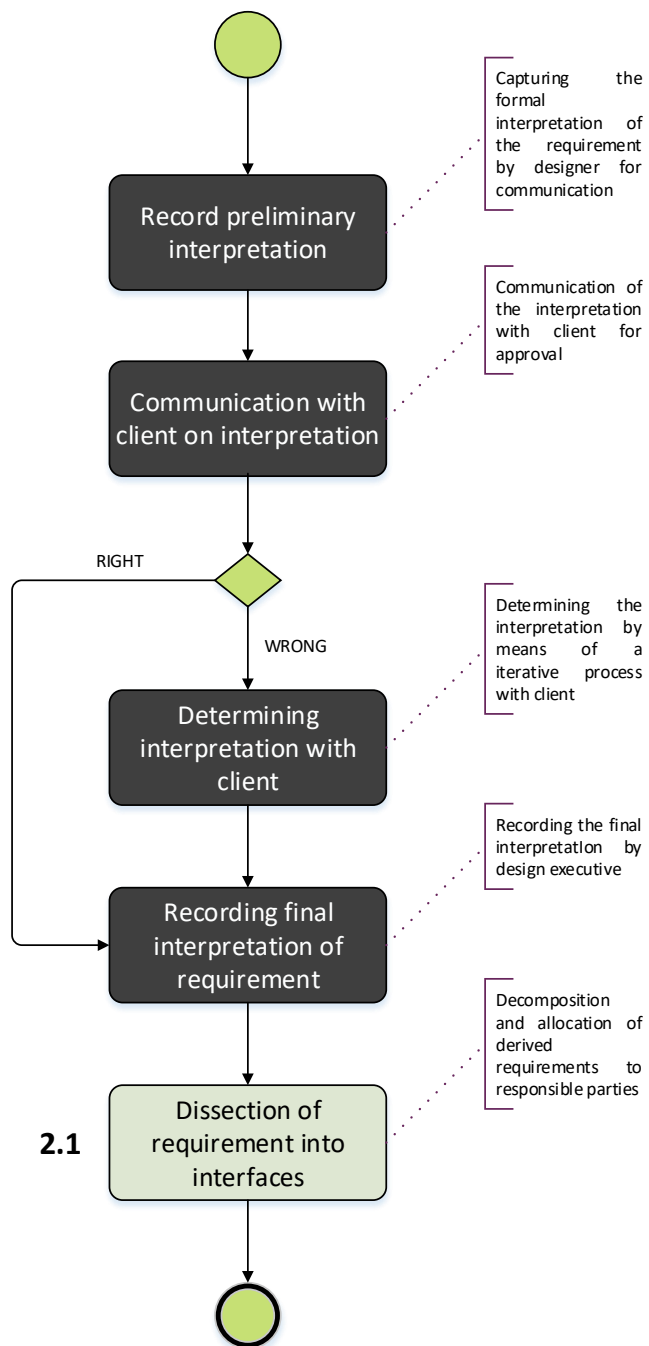


Figure 14: Activity diagram 2: Assessing requirements on SMART principle.

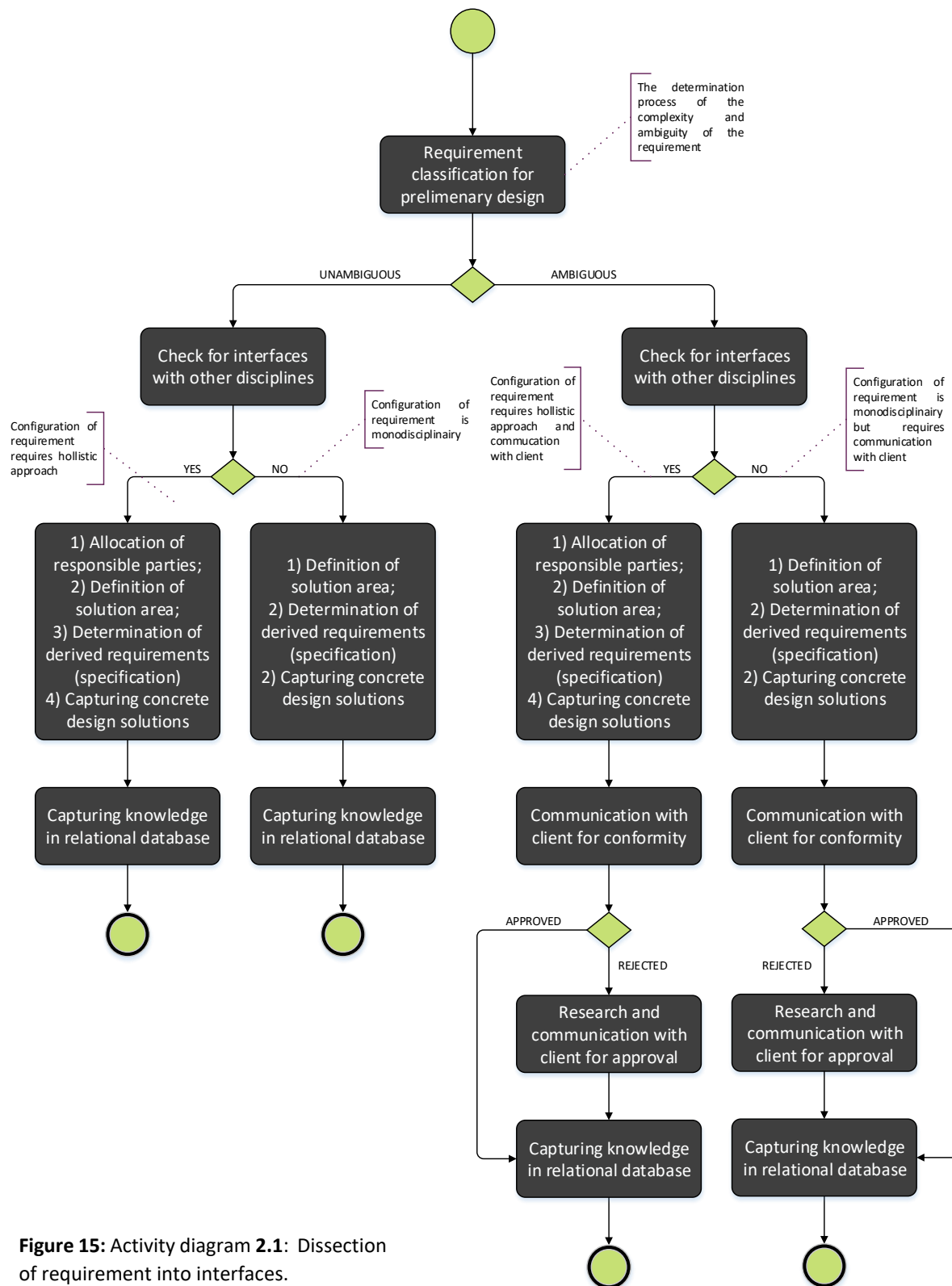


Figure 15: Activity diagram 2.1: Dissection of requirement into interfaces.

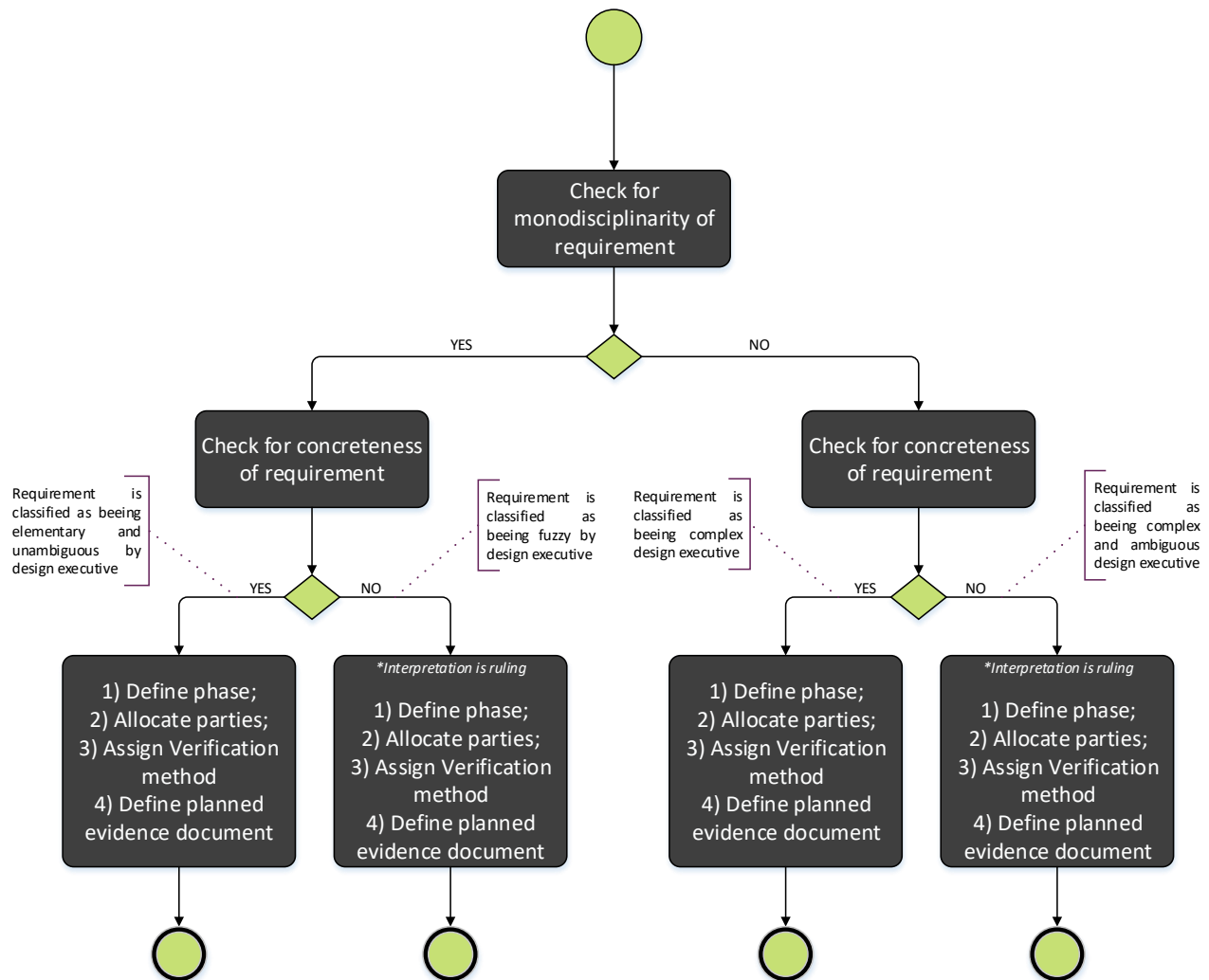


Figure 16: Activity diagram 3: Verification planning

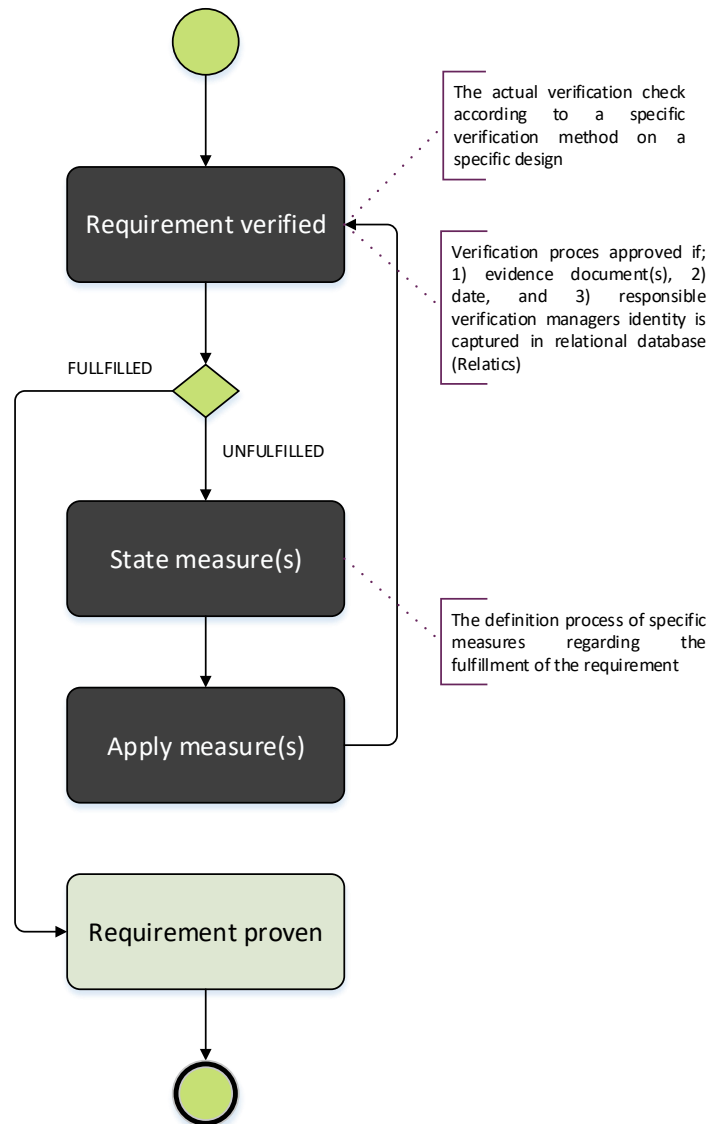


Figure 17: Activity diagram 4: Verification

4.4.3 Verification

There has been found that the quality of verification depends on the process that takes place prior to the requirement being finalized. Verification is only deemed relevant if the steps after the verification procedure are defined properly. This basically implies that preconditions are required to be met before verification can start. The answers obtained from the interviews, made it possible to define the preconditions for the total verification process: the interpretation of the client specific requirements must be communicated and validated by the client; the allocation of requirements to elements must have been done in a sense that all the elements which a requirement is applying to are allocated and captured; the definition process, of whether a verification complies or not, must be done in a way that an unambiguous answer can be given as a response to the satisfaction of the initial requirement; the definition of the LOD of a design, allocated as a function of time according to the level of risk, must be done adequately; requirements which are assumed to contain a higher level of risk should be monitored very closely since this could prevent the discovery of defaults in later stages; the level of detail in regards to different disciplines need to be defined clearly to validate the verification due to the fact that different levels of detail come from different disciplines which can instigate problems by revisions.

The core elements of the verification process are closely related to the definitions as laid down within the design process. The core elements of verification is to construct the right comparisons between building information and building model. This implies that the comparison of a requirement and object must be executed correctly and defined unambiguously. This implies the following steps that correspond with the actual process: *preparation*: the definition of the defined elements as used in the verification must be unambiguous. This is mostly realized in the design phase where the allocation is defined; *defining the verification plan (procedure)*: the process of structuring the right procedure and assigning the right rules of verification; *verification during the design process*: the verification can be executed and documented after the verification procedures are defined and the elements are designed.

4.4.4 Requirement classification

There has been noticed that classification of client specific requirements has multiple definitions. A clear difference between the following three types of requirements is defined according to the following distribution: value (numerical) requirements; relational requirements; textual requirements. As described within the problem definition of this research initiative, textual requirements often are difficult to handle since they are sensitive for misinterpretations. Besides the classification in measurability, also difference by interpretation among the various disciplines within the AEC industry has come forward. Technical requirements are known to be different than architectural requirements. These requirements are known to be mostly related to different qualities of a building. For instance, technical requirements are known to be related to comfort requirements, whereas architectural requirements are more often related to aesthetics. It is generally assumed that a requirement will be marked as complex whenever it is not tangible or measurable.

4.4.5 Automation of translation procedure

The initial (automated) translation procedure for the translation of requirements into product specifications is not researched upon that greatly by researchers from the AEC industry, especially in case of non-functional requirements. The interviewees emphasized that it is hard to translated physical and functional requirements by means of automation, given the fact that information on design decisions from the past have never been captured by means of standards and semantics. The decisions from the past are not usable given these circumstances. This makes it even more challenging to initiate a first attempt to automate this process for the class of non-functional requirements. The interviewees pointed out that it is very difficult to automate the translation procedure of non-quantifiable, more qualitative requirements (look and feel requirements) given the previous reasoning. There are no standards in what quality requirements can be compared to according to knowledge as gained from previous projects. Among the interviewees, having a system that could prompt users with experiences from the past, by the dissection of non-functional requirements, is assumed to be very useful as a support tool by decision-making during the design process. It is assumed that this could improve the process of client / designer interaction. A very important pre condition for using information by the design of such automated system is that it should be able to use input information of verified and validated projects from the past.

The preconditions for a system that automates the requirement translation procedure have been interviewed upon. There have been several trials and demonstrations in which attempts have been initiated to automate this procedure. There has been tried to interpret text from client's briefs by use of semi-automated techniques to enrich this information in terms of SE information and data to support the SE process. These attempts were unsuccessful given the following reasons:

- 1) Specific knowledge on lexical analysis, requirement ontologies, standards and semantics are missing on an organizational level;
- 2) The organization is used to build after design, rather than design and build which results in a lack in functional design competences;
- 3) Yet, there is no standard in which concepts are captured, there is no formal language in which requirements can be defined and interpreted by both human and computers;
- 4) Information and data is not delivered nor captured by means of a standard structures or file formats.

There is found that this process could be automated if the following input can be provided to feed such a system:

- 1) Validated interpretations of requirement with client and users as obtained from previous projects;
- 2) Availability of a set of dissected requirements, as programmed in previous projects, that are represented in a measurable state;
- 3) Allocation of requirements and objects, as done in previous projects;

- 4) Availability of the total amount of information in regards to the verification procedure, as executed within previous projects.

This implies that the total process which happens before verification, as obtained from previous projects, is captured in a clear and consistent way. If this is done, then it could be possible to consult this information to ground reasoning by decision making during design stages. The definitions of requirements and corresponding verification method should be clear to safeguard that the valid data is used as knowledge. This could also prevent a lot of rework by each project, given the fact that reasoning by decision making is captured by means of standardization that can be reused.

The interviewees emphasized that it can be very hard to capture each and every single decision by the translation of non-functional requirements by several reason. The first reason is due to the fact that the AEC industry is working pragmatic rather than systematically. There has also been found that the tight planning in which requirements analysis are planned is not offering the possibility to do this in detail, especially in case of bigger complex projects with numerous amounts of non-functional requirements. This implies the need for a certain information system that can be consulted within the early design stages to consult for specific queries. This system could provide knowledge on what decisions have been taken by the translation of certain non-functional requirements from previous projects.

The eventual way for designing this automated translation system is also questioned upon within the interviews. These fundamental system requirements are captured as follows:

- 1) The fundamental operational functions such as Create, Read, Update and Delete (CRUD), need to be accommodated in the system;
- 2) The system needs to be capable to run automated lexical analysis to dissect the linguistic chunks of text, in which product requirements are listed, within a client's brief;
- 3) The system needs to be able to store enrich words, as obtained from the dissected text, by means of state of the art knowledge to formulate the possible meaning(s) of the word within a sentence;
- 4) The systems needs to be able to allocate the enriched words in relation to the subsystems by means of a standard system distributions, such as the NL/SfB;
- 5) The system needs to capture and distinguish the translation of the definitions of obtained words in relation to other acting disciplines, assuming that these definitions can vary;
- 6) The system needs to capture the final specification of the word(s) to formulate a product specification that satisfies the initial requirement;

Chapter 5 will treat the evolutionary prototyping process according to this fundamental set of system requirements.

This page is intentionally left blank

5 Model

5.1 Method

5.1.1 Evolutionary prototyping

This paragraph addresses the fundamental reasoning for the selection of a certain prototyping method. Numerous researchers have contributed to knowledge regarding the variety of prototyping techniques. This research has introduced both a literature review and an investigation on in-house practices to explore the required knowledge for the development of the desired system, and to derive and measure the client's needs for the development of a certain system in which requirements can be interpreted, translated, allocated and specified. During the early stages of this research initiative, there has been found that the interviewed experts were very interested in the development of such system. On the other hand, there has been found that their detailed requirements for the development of such a system were yet fuzzy. The operational functionality of the desired system was clear, however, specific requirements for detailed software development were in default. The in-house practices functioned therefore as the instrument to measure the necessity of such a system within the business process, and what the fundamental requirements of such system should be. This was an iterative process which evolved over time.

The detailed client requirements weren't clear enough at the beginning of this research initiative, which made it hard to introduce static prototyping technique(s). Therefore, the in-house practices functioned as a great mean to discuss, collect and evaluate client requirements regarding software development. The clearance of the requirements of the operational functionality of the system was one of the main reasons to introduce evolutionary prototyping over other prototyping techniques such as 'Throwaway/Rapid prototyping' or 'Extreme prototyping'. The fundamental definition of an evolutionary prototyping approach can be explained by the incremental development of software where an initial prototype is produced and refined through a number of stages, before achieving the final system (Beaudouin-Lafon, Mackay, 2007).

This approach is very pleasant to introduce where client requirements are fundamentally clear from nature, but somehow fuzzy in detail. The clearness reflects in the operational quality that the system should accommodate; the fuzziness reflects in decisions which yet need to be made according to trial and error. By this approach, parts of the systems can be developed and optimized by means of the interaction between developer and the field experts. Then, additional system requirements could grow as a function of the development process. Hereby, numerous of additional system requirements can be derived and formulated, and significant parts of the system can be developed to design the core concept of the system. This also gives the client the opportunity to be more influential during the system development stages which verifies whether the most recent developments are matching the desired outcome.

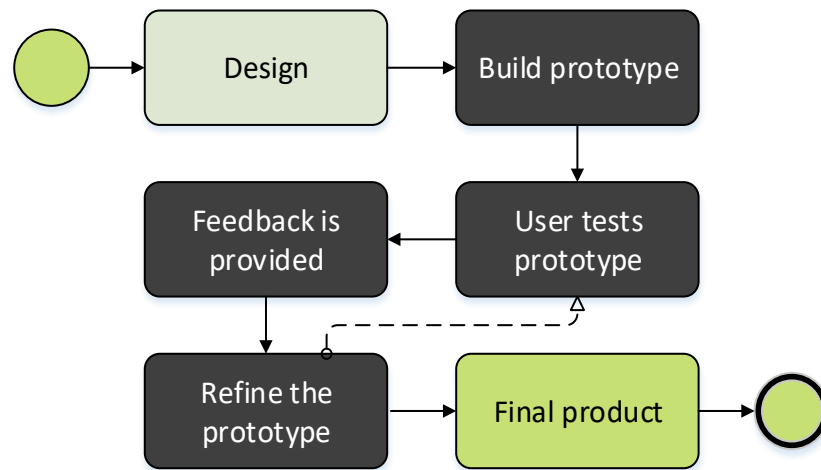


Figure 18: Evolutionary prototyping process.

Within this development process, the evolutionary prototyping process has been dissected in 6 parts. During the *Design stage*, the client requirements are interpreted and verified with the client. These fundamental system requirements are the foundation for the development of the model. The first version of the prototype will be built within the *Prototyping phase*. Here, the initial client requirements are accommodated within the functionality of the system. This system will be introduced to the client for a *User test* of the prototype. The client will use, analyze and evaluate the delivered prototype and report the feedback. After the obtained customer's feedback, the revision of the model can be made within the *Refining stage*, then the prototype can be delivered to the client in order to execute a User test gain, this is an iterative process until the functionality of the system has been achieved. Whenever the desired system performances are achieved, then the *Final development* can be started. Even during this stage, the client might state additional requirements. In this case, the development loop will be introduced again to refine the system its functionality. After the *Final development* phase, there is assumed that the functionality of the system is matching the desired requirements of the client.

Generally speaking, a prototype is used during its development. The prototype is not used only for feasibility analysis or for evaluating other types of system requirements, but also for the creation of user interfaces. This part needs to be emphasized within this paragraph given the fact that this system needs to be very pleasant to deal with by the layman. User interface (UI) prototyping is an iterative development technique in which users are actively involved in the design of the UI of a system (Beaudouin-Lafon, Mackay, 2007). However, the user interface prototype is built early, before the whole system was analyzed, designed and implemented (Beaudouin-Lafon, Mackay, 2007). This gives the client a good understanding of what it desires. Therefore, user interface prototyping has also been introduced before the prototyping process. This could measure and optimize the user experience whenever consulting the system. This process was structured in two stages, given the mock-up stage and the implementation of the mock-up stage. The *mock-up stage* was very useful. This process revealed what the system should look and operate like by means of client / developer interaction. The *implementation of the mock-up stage* was introduced to

integrate all of the additional requirements that were obtained from the mock-up stage. This process has been executed previous and parallel to the iterative prototyping process of the system itself, wherefore numerous additional requirements have been obtained.

5.1.2 System requirements

As described within paragraph 5.1.1, the system is developed by means of an evolutionary prototyping approach. This due to the fact that the fundamental operational functionality of the desired system were captured by means of the in-house practices that provided the possibility to interact with field experts to optimize and introduce additional system requirements.

The fundamental system requirements that are captured and used for the initial development stage are as follows:

- 1) The fundamental operational functions such as Create, Read, Update and Delete (CRUD), need to be accommodated in the system;
- 2) The system needs to be capable to run automated lexical analysis to dissect the linguistic chunks of text, in which product requirements are listed, within a client's brief;
- 3) The system needs to be able to store enrich words, as obtained from the dissected text, by means of state of the art knowledge to formulate the possible meaning(s) of the word within a sentence;
- 4) The systems needs to be able to allocate the enriched words in relation to the subsystems by means of a standard system distributions, such as the NL/SfB;
- 5) The system needs to capture and distinguish the translation of the definitions of obtained words in relation to other acting disciplines, assuming that these definitions can vary;
- 6) The system needs to capture the final specification of the word(s) to formulate a product specification that satisfies the initial requirement;

The fundamental system configuration, operational functionality and graphical user interface are obtained iteratively.

5.2 Use Case(s)

5.2.1 Use case 1: TRANSLATE

The initial use case of this system can be explained by its allocation as a function of the business processes. The use, allocation, system requirements, and the functionality of the desired system has been captured within section 4.4. To be more specific, the following use case(s) and use case text(s) that are presented within the upcoming paragraphs have been developed prior the development stage. The first use case, TRANSLATE, is presented in Figure 19 and Figure 20 beneath. This use case diagram communicates the use of the final systems by its user for translating a client specific requirement into specification a product specification.

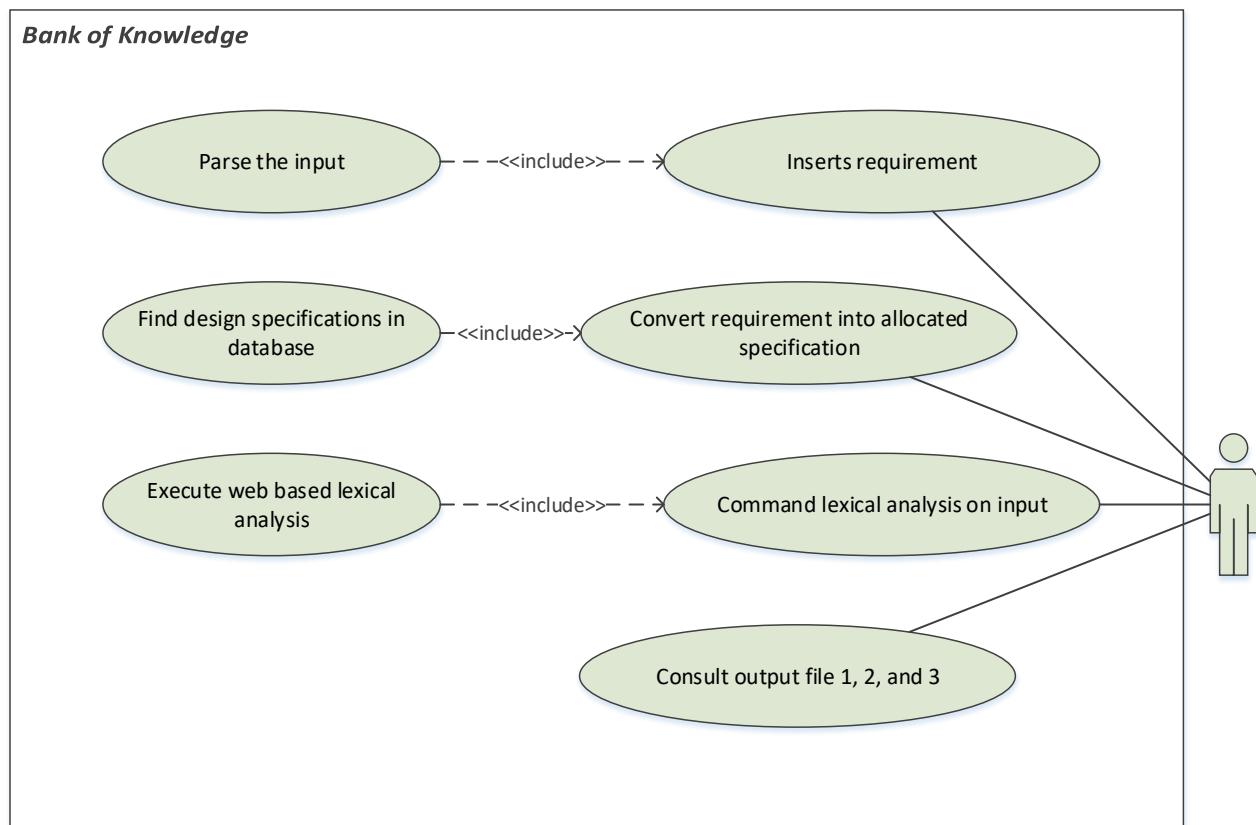


Figure 19: Use case 1, use case diagram: TRANSLATE, Bank of Knowledge.

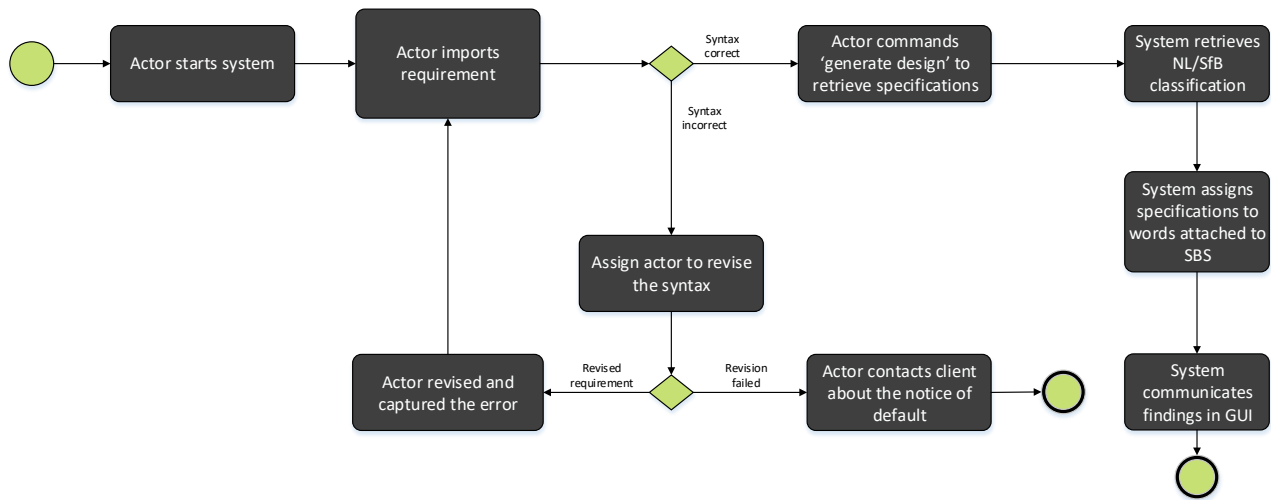


Figure 20: Use case 1, Activity diagram: TRANSLATE, Bank of Knowledge.

Use Case	Translate
Actors	Designer - Engineer - System Engineer
Preconditions	Designer - Engineer - System Engineer is licensed and familiar with the expert system
Description (standard path)	S1. User open the system S2. User imports the requirement S3. User commands 'generate design' to retrieve specifications S4. System retrieves NL/SfB classification S6. System assigns specification to words attached to SBS S7. System communicates findings in GUI
Postcondition (result)	The KBS has enriched the requirement with state of the art knowledge; assigned this information and data to a discipline; and proposed specific requirements in relation to the discipline as a function of the initial requirement
Extension	-
Exception	The requirement its syntax and semantics are in default; the actor can not start its activities
Alternate path	A01: @S2. Actor imports the requirement A01.02 Prompt actor(s) empty output screen whenever the syntax is incorrect A01.02 Actor checks the syntax of the requirement A01.03 RESUME @S02 if analyzed syntax and semantics are revisable A01.04 Actor contact the client about the notice of default
Postcondition (result)	The client has been sent a notice of default since the requirement has contains syntax errors

Table 5: Use case 1, Use case text: TRANSLATE, Bank of knowledge.

5.2.2 Use case 2: Database Manager BOK

The second use case of this system is related to the activities that contribute to create, read, update, and delete Tokens within the system its database. The database, data format, and the formal language that is implemented to CRUD information and data within the database will be treated within chapter 6. This second use case, CRUD, is presented in Figure 21 and Figure 22 beneath. This use case diagram communicates the use of the final systems by its user, the Database Manager(s) of the Bank of Knowledge (KOB), for specific activities that are related to the systems database management.

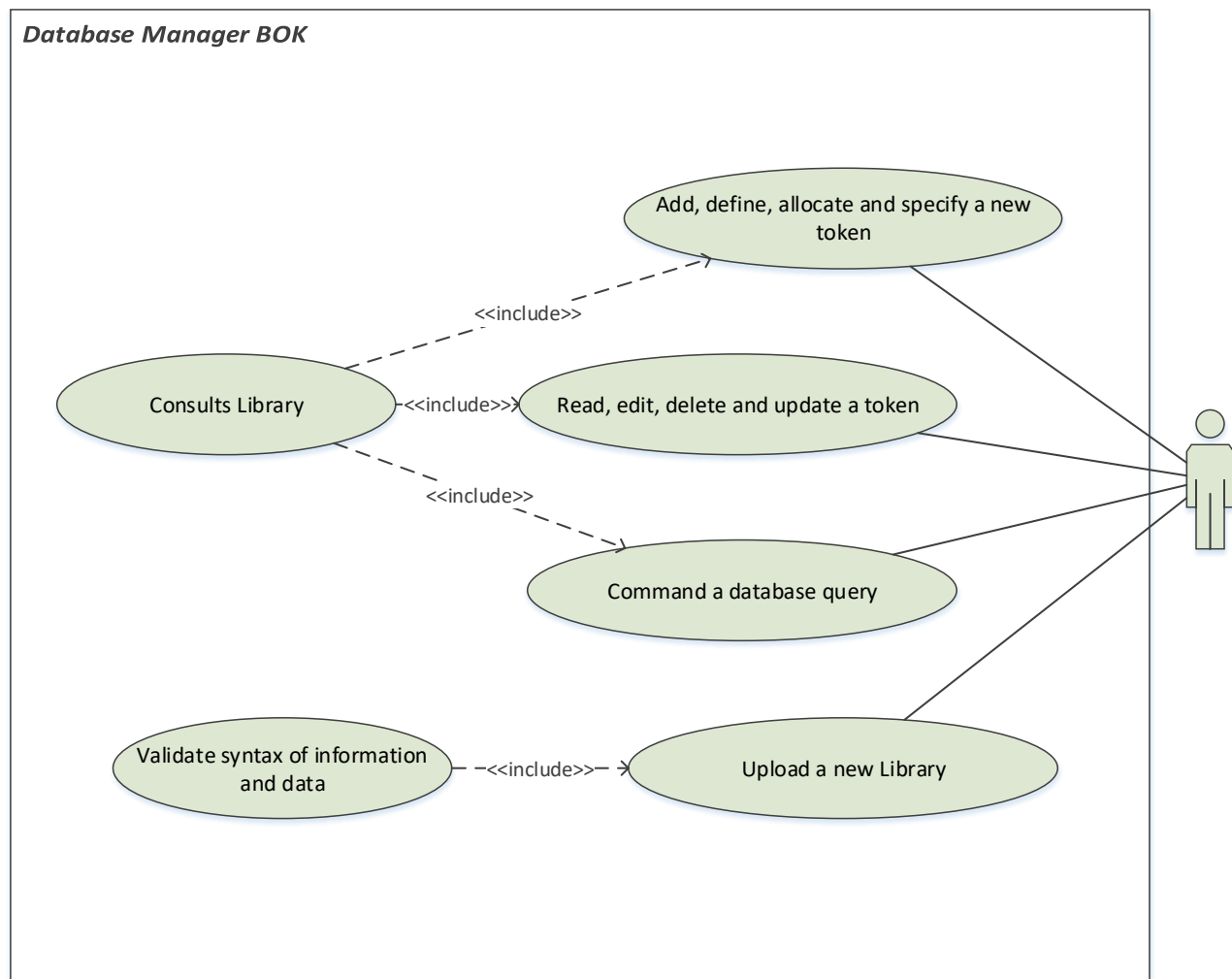


Figure 21: Use case 2, use case diagram: Database Manager BOK.

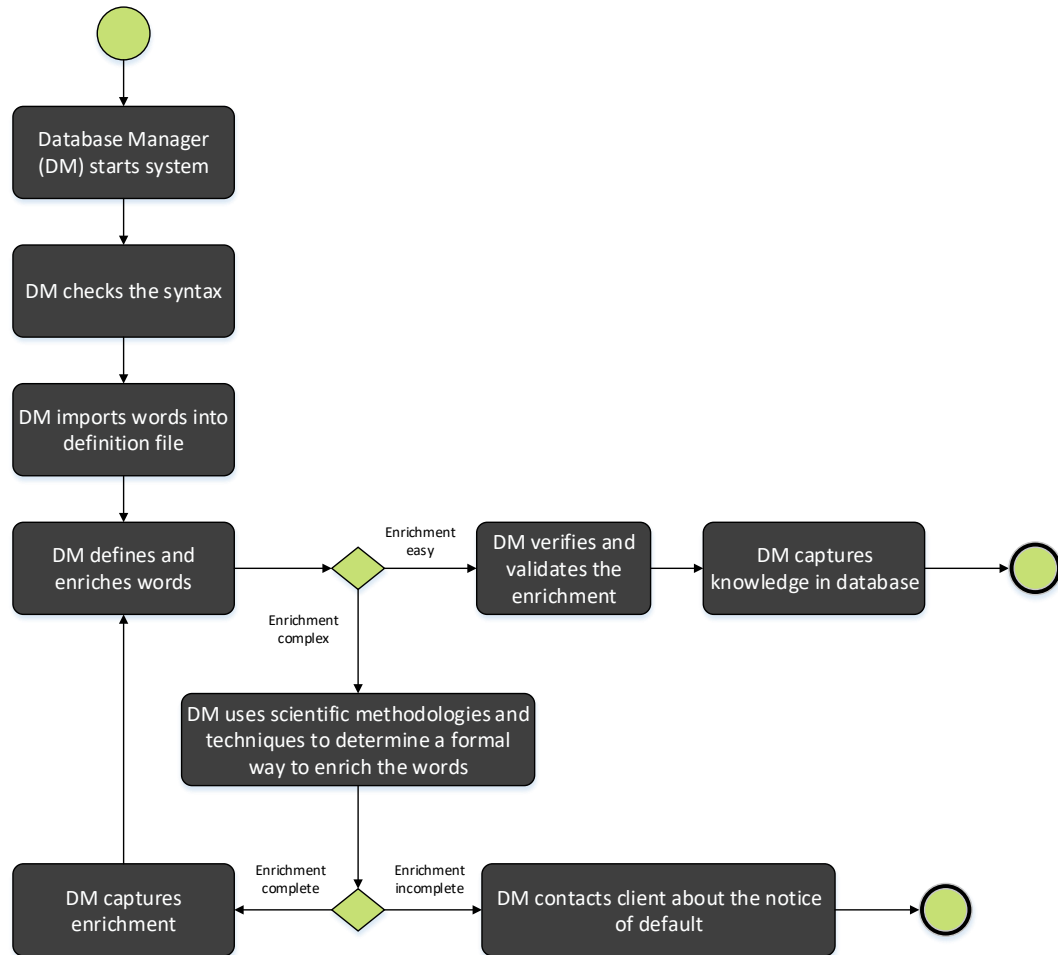


Figure 22: Use case 2, Activity diagram: Database Manager BOK

Use Case	Database Manager BOK
Actors	Database manager (DM)
Preconditions	Database manager is licenced and familiar with the expert system
Description (standard path)	S1. Database manager opens the system S2. Database manager checks the syntax S3. Database manager imports words into the definition file by commanding 'open definition file'* S4. Database manager defines and enriches word in terms of constraints specification in architecture and by, if possible, information and graphical representations by use of (valid) links from the World Wide Web S5. Database manager verifies and validates the enrichment S6. Database manager stores and updates the captured knowledge in the CSV file by commanding 'open definition file'
Postcondition (result)	The system its (growing) database is enriched by means of knowledge
Extension	-
Exception	Actor(s) cannot define nor enrich the word(s) given uncertainties
Alternate path	A01: @S4. Actor uses scientific methodologies and techniques to derive a formal way to enrich the word A01.02 enrichment complete Resume @S5 A01.03 enrichment impossible, actor contacts client
Postcondition (result)	Database is upgraded with actual and valid information and data

Table 6: Use case 2, Use case text: Database Manager BOK

5.3 Prototyping process

Due to the iterative nature of the implemented prototyping methodology and the corresponding technique to report these process, there has been chosen to summarize the prototypical developments that have evolved during this development process. This reveals the incremental growth of the system as a function of client/developer interaction. The phases in which versions of the prototype, and their corresponding operational functionality, have been designed and produced are recorded and listed below.

First prototype versions (alpha version)

- The graphical user interface (GUI) consisting of:
 - an input field where requirements could be formulated as obtained from the client's brief;
 - a command button named as 'generate design' where the user can start a enrichment process of the words as obtained from the requirement;
- The result of these enrichment procedure was outputted in a text file.

Second prototype version (beta version 0.0)

- Here, adjustments have been introduced within the GUI of the system due to client feedback as a function of UI prototyping:
 - Three command buttons were added, given:
 - Open output file, which opens a generated output in a file;
 - Open definitions file (database), which opens the system its database by means of a CSV file format to be able to CRUD the captured knowledge and information;
 - Reload definitions, to reload and update the extended knowledge as captured by the CRUD principle within the system its database;
- Still, the result of the enrichment procedure was outputted in a text file. However, the data in this file was then displayed on the output field within the GUI.

Third prototype version (beta version 1.0)

- Significant changes have been introduced in the GUI:
 - Three output fields have been integrated in the GUI instead of a single output field:
 - The first one is used for the definition of a word;
 - The second one is used for the word as defined in types;
 - The third one is used for the values connected to this word (e.g. a product);
 - Two command buttons were added and one has been changed:
 - The open output file command 1, 2 or 3;
- The result of each output was first captured in a text file, and then displayed on their output fields.

Fourth prototype version (version 1.7.0)

- Slight changes in the GUI interface:
 - One command button is added:

- The lexical analysis command button which will execute an automated web based lexical analysis.

Final version (version 2.0)

- An editor, Database Manager Bank of Knowledge (BOK), for the definition file (database) was added. This editor contains the following elements:
 - Four buttons to add html elements to the definitions;
 - A token panel consisting of the following elements:
 - An word input field;
 - A definition panel, containing all (class) definitions and specifications of the word from the input word field, and a button to add a new (class) definition or specification to the word;
 - Five buttons:
 - Edit token:
This button sets the word of the token selected in the database to the token in the input field and the (class) definitions and specifications to the definition panel.
 - Save token:
This button saves the token to the local library. You can now use this token in the advisor application. Note that this does not permanently saves this token.
 - Save changes:
This button saves the local library to a file. Note that this does not save an unsaved token.
 - Del selected def(s) and Delete token:
The Del selected def(s) delete all selected (class) definitions and specifications in the definition panel. The Delete token deletes simply tokens in the database.
 - A database panel consisting of the following elements:
 - A search bar with search button:
Here the user can add its search term to search for the word of a token in the token selection panel. It uses a simple search algorithm that only compares the search term with the (first part of the) word of each token. Pressing the search button or enter will both start a search. To reset, enter nothing and press enter or search;
 - A token selection panel containing the tokens from the (local) database. A token can be selected by clicking on it, but at most one token can be selected at once. Using the search function can reduce the number of shown tokens.
 - When the editor is started, it copies the definition file to a backup file. Since the editor only hides from the user when it is closed, it will only make one backup file per session. Up to nine different backup files can be stored.
- Some changes in the GUI interface of the advisor:
 - The reload definitions button was replaced with the Open editor button which opens the editor; a show/hide button for the word enrichment panel to show/hide this panel.

5.4 System operational functionality

The following descriptions, which are summarized for the sake of brevity, ground the operational functionality and reasoning by system design:

Short description of the operation of the program at startup:

- Create the GUI of the advisor application;
- Read the tokens from the definition (database) file.

Short description of the operation of the program for every command button in the final version of the program:

- Command button: Generate design
 - Commands the program to read the text in the input field and stores this;
 - Splits the text into sentences, and the sentences into words;
 - Iterate over every single word by checking its definition in the preloaded definition file (database);
 - When a word was found in the database, then the program list its definitions in the three output files;
 - When finished with parsing all words, then the program reads the three output files and displays their contents in the three output fields within the GUI.
- Command button: Open output file [1, 2, 3]
 - Directly opens the output file 1, 2 or 3.
- Command button: Reload definitions (not in final version)
 - Throws away the previously stored old definitions as captured within the definition file (database);
 - The program parses the updated definitions of the definition file (database).
- Command button: Open editor
 - Opens the token editor.
- Command button: Open definition file
 - Directly opens the definition file in CSV format which is basically the database within this system.
- Command button: Lexical analysis
 - Parses the text in the input field and stores it;
 - Generates a link to a lexical analyzer that will directly process the text by means of a web based automated lexical analysis (Noah's Ark group, 2017);
 - Opens the link in the default web browser.

Declaration on database structure in CSV format:

- Every row within the CSV file defines one word;
- Empty rows and all text (including other cells) after “/:\” are ignored by parsing;
- In the first cell in the row is the word to be defined;
- In the other cells are definitions of that particular word. Every definition is put in a single cell and contains two properties, given:
 - *NL-SfB class*: denotes what type of definition it is. Here, default is “other”, which denotes the lexical definition of the word. To change the class, simply write “*<NL-SfB class>” (where “class” is any of the predefined classes) in any cell before the definition. When using multiple class changes, only the last one before the definition counts;
 - *Specification*: denotes whether the cells contain a definition or a value. To change the specification, simply write “**<specification>” (where specification is either “value” or “def”). Here, default is “def”. When using multiple specifications changes, only the last one before the definition counts;
- All definitions within the database can be enriched using HTML. To add, for example, a link in HTML, write: source , where ‘xyz’ is the link and ‘source’ is the linguistic description of the link.

Short description of general choices:

- There has been chosen to introduce a separated definition file instead of preprogrammed definitions since these are then easier to alter from outside the program as a function of time;
- Generate output files instead of directly putting the data to the output fields to create the possibility to use the information and data outside the application;
- A hash table is introduced for linking the words with their definition. This decision relies mainly due to the fact that a hash table has the fastest ‘add’, ‘remove’ and ‘get value’ operations possible;
- Multi-threading is used to parse the definitions to avoid hanging the application. This means that multiple parts of the program are executed at the same time. However, this will cause multiple hard-to-trace bugs if not programmed carefully.

Here, a brief introduction has been given on the operational functionality of the program to make this development process understandable to the layman. The following paragraph will explain the reasoning by system design in more depth.

This page is intentionally left blank

6 Results

6.1.1 The Bank of Knowledge application

This section treats the results as obtained from system design as described within chapters 6 and 7 of this report. The operational functionality of the Bank of Knowledge will be treated within this section. The aim of this chapter is to provide an overview of how this research initiative have resulted to the development of the program. There has been chosen to report the results of this tool chronologically, since this matches the procedure for both use cases of this system the best.

6.1.2 The Graphical User Interface

The main emphasis of program development relied on the fact that this program needs to be easy by use of the layman. This implies that the user interface and the accommodated functionality of this program is required to be user friendly, and therefore easy to understand by untrained people. The main aim of this part of system design reflects in a graphical user interface where all functionalities are directly accessible by the users. The main functionality of this tool are described in section 5.4 and are demonstrated in section 7. The illustration beneath represents the Graphical User Interface of the Bank of Knowledge where the mean functionality of the system is accommodated. This is done as a preparation for further descriptions on system results within this chapter. Its aim is to provide practical insight in the configuration and operational functionality of the tool. This resulted in the following configuration:

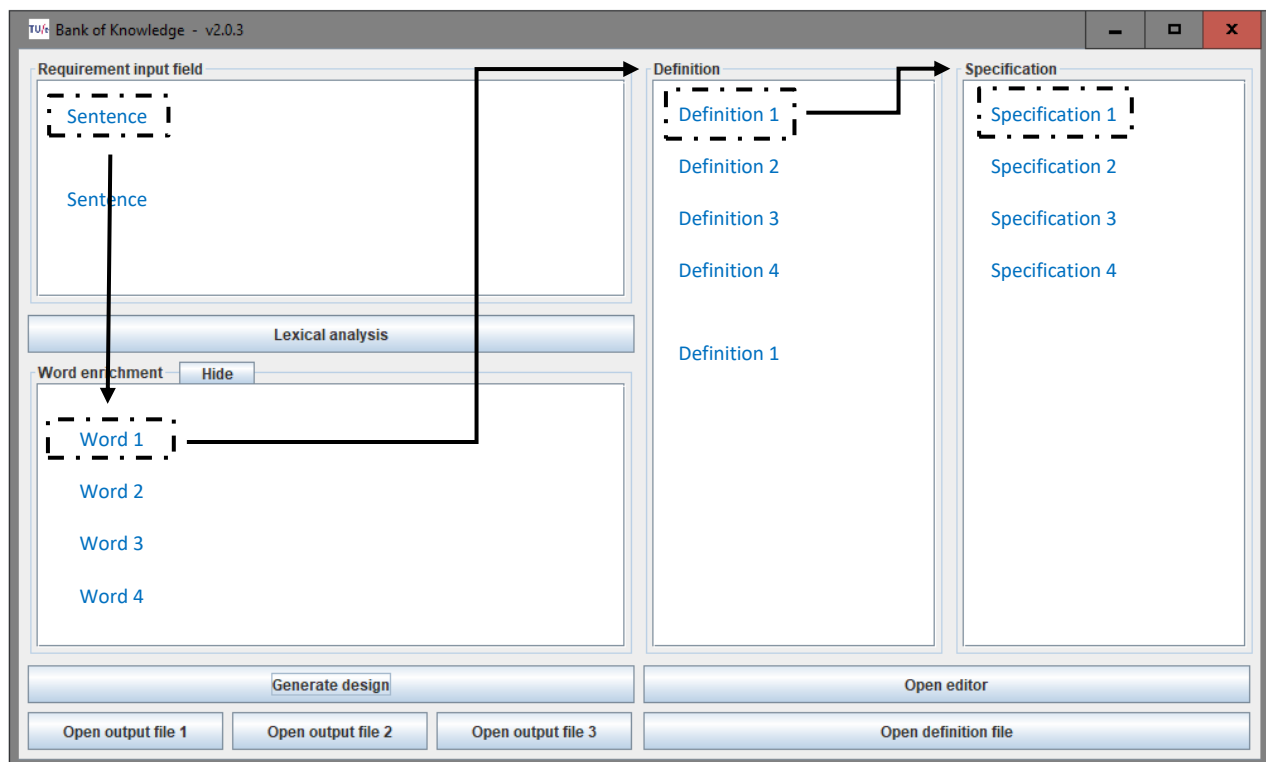


Figure 23: The graphical user interface of the Bank of Knowledge system (Use case 1).

6.1.3 Lexical analysis

The fact is that requirements as obtained from the client's brief reflect in mutual contractual obligations. According to the UAC-IC 2005, the client is responsible for presenting and communicating these client specific requirements in a clear and unambiguous way. On the other hand, the designer has its obligation for the duty to warn. This, not only to regulate business processes, but also to safeguard the relation between a designer and client as a function of business processes. This is very crucial given, since the designer is responsible to communicate defaults in the specification of requirements towards the client. Designers and engineers are known to be specialized in designing and engineering rather than linguistics. Therefore, it might be very important to be able to decide, in an effective and efficient way, whether a requirement is ambiguous. This current determination procedure on whether a requirement is ambiguous or not is currently done fully manually, which is very time consuming and error prone due to the possible misinterpretations of the reader. Therefore, there has been decided to integrate an automated web based Lexical Analyzer which dissects the sentences by means of a Syntactic Dependency parse and a Frame-Semantic Parse (Noah's Ark group, 2017). This is way more effective and efficient in comparison to the traditional laborious client requirement interpretation procedures. By this technique, words can be characterized by their function within a certain sentence. Then, observations due to the first layer of ambiguity can be addressed.

6.1.4 Word enrichment

Requirements are known to be written in chunks of text that contain linguistic definitions and properties. Interpreting and translating a single word, or a relatively small sentence seems to be more effective if executed manually. However, in the construction industry, client's brief can possibly contain thousands of requirements. These linguistic descriptions are known to be described by a client's preferences, rather than standards in terms of syntax and semantics (CROW, 2017). This can possibly instigate a layer of ambiguity during information exchanges. It is very important to demarcate the interpretation of words in a sentence, in order to demarcate the interpretation of a whole sentence. Therefore, there has been chosen to translate the variety of words as obtained from certain client specific requirements into enriched words where the designers and engineers can start working (safely) with. This procedure is also very crucial since here, the origin of the interpretation and definition(s) of the words and sentences will be captured. The knowledge or information that can be attached to a certain word are very flexible from nature. This flexibility rises from the possibility that an interpreter (D&E) can determine the definition of the word. There has been chosen to capture the definitions by means of state of the art knowledge that is offered by literature. Nowadays, most literature is digitalized or easy accessible by means of the World Wide Web.

Therefore there has been decided to introduce the use of knowledge and information as obtained from the World Wide Web. The classification of which type of knowledge or information that there can be attached to a certain word is very flexible in nature. However, this research initiative holds on a certain formal format that is integrated in the current data format of the database, which will be treated in more depth within paragraph 6.1.7. The D&E are more than free to reason their

interpretation by means of the state of the art knowledge as obtained from the World Wide Web. The fundamental preconditions by the classification of the definitions for interpretation by the D&E can be categorized as follows:

- 1) The precondition is that the sentence, that consists of words, has been analyzed by the automated lexical analysis plugin;
- 2) The word needs to be defined by means of state of the art knowledge as obtained from valid literature as gained traditionally, or from valid literature as obtained from the World Wide Web;
- 3) The obtained definition of the word needs to be characterized, classified and captured by means of its source. Here the flexibility comes in play, since the D&E are more than free to attach definitions of words under the fundamental condition that it can be reasoned by valid state of the art knowledge and or semantics;
- 4) The obtained definition of the word needs to be allocated according to a certain formal system that relates sub systems as a function of a system. This makes it possible to allocate requirements on an object level;
- 5) All of the previous steps need to be monitored and logged carefully within the definition file of this tool which is the database that will grow as a function of time.

6.1.5 Word allocation

The previous paragraph 6.1.4 has been used to introduce how the words that are enriched as obtained from the sentences can be captured. These sentences are the client specific requirements as obtained from the client's brief. This technique can be used to enrich both the interpretation and definition domain of a requirement. This procedure is from great importance by the interpretation of the correct meaning of requirements as captured during the briefing stage. Yet, another very important aspect successive to this procedure is the allocation of client specific requirements on an object level within a system. Here, the link between requirements and objects will be realized. However, it is a tough job to do this, because words can possibly vary in their definition. This simply by the fact that a word can have different meanings within different domains. It is important to determine and justify for what purpose the word has been used. This process is very crucial by decision making based on decision documents that are written natural language. For example, the definition of the word 'warm' can have a different definition domain in terms of 'architectural amenity' than the design of a 'heating system'. The definition of the word 'warm' could instigate ambiguity between designing parties, this especially for the allocation of requirements (that are built from words) to objects on a subsystem level. The following principle is introduced within the functionality of the program to safeguard and regulate this procedure:

- 1) The interpreted words are defined and enriched with their possible meaning;
- 2) The requirements are analyzed according to their application on an object level;
- 3) The analysis on the application of a sentence on an object level have been related to subsystems, according to a formal system distribution (NL-SfB in this case) where this word is most likely to apply to;

- 4) Based on the findings of step 3, the words are defined in terms of their meaning regarding to the acting discipline and the object within the SBS that is affected by its application;
- 5) The definition of the word is captured in a sense that declares ambiguity among acting parties, based on the methods and techniques as treated within 6.1.7, regarding a certain application of a sentence on an object level.

6.1.6 Requirement specification

Whenever a requirement is interpreted, enriched, and allocated on an object level, then the possibility of specifying a requirement arises. This implies that specifications can be formulated from the pre-treated requirements. This makes it possible to translate the actual application of a certain requirement on an object level in terms of the final design decision; the product specification. This is a crucial process during the translation procedure of client specific requirements into product specification, especially in case of non-functional requirements. During this activity, it is known to be essential to derive all mono-disciplinary requirements that a non-functional requirement can contain for a certain disciplinary application on an object level. This could possibly contribute to formalization of a certain specification. Yet, non-functional requirements are known to possibly contain multiple interfaces that are integrated within a single requirement. It can be very hard to distill mono-disciplinary requirements. Here, the definition of a mono-disciplinary requirement is assumed to be a requirement where one acting party is responsible for (e.g. an architectural, or structural, or heating requirement). The harmonized fusion of a set of mono-disciplinary requirements are assumed to contribute to the holistic configuration of a complex non-functional requirements.

Problems during the translation of the physical and functional requirements are not that error prone as by the translation of the non-functional requirements. These are often non-quantifiable, and more qualitative from nature. They are not that tangible and measurable as required for instant translation into product specification during early design stages. To be more specific, value (numerical) requirements and relational requirements are easier to interpret and specify than textual requirements. This implies that textual requirements need a certain treatment before they can be specified. This is basically the essence of the operational functionality that the program will offer as a function of time. The system is able to store the reasoning during the dissection procedure of a requirement into a product specification which can be consulted for future translation procedures in upcoming projects.

The previous steps as described within this chapter, that are proposed for dissection, interpretation, enrichment, and allocation of a requirement contribute to the possibility to specify a qualitative- textual requirement. The treatment of a qualitative- textual requirement is such that it needs to be dissected from its interfaces with other objects within a system. For instance, if a requirement implies *“that a room needs to withhold a cozy atmosphere”*, then there needs to be questioned and reasoned how this affects the interfaces between objects on a system level. More specific, what measures need to be taken in terms of architectural, structural, MEP design? And

how can these mono-disciplinary requirements be distilled from this qualitative- textual requirement?

The methodology that is accommodated within functionality of this program provides the opportunity to determine the meaning and application domain of a certain qualitative- textual requirement before merging them. This will be discussed in chapter 6.1.7. This provides the opportunity to harmonize interfaces that the requirement can imply; before capturing its holistic meaning. This dissection process aims for measurability according to the structural composition of a requirement which provides the possibility to determine the specification of a non-functional requirements.

6.1.7 Database and input

This paragraph functions as a practical description on the operational functionality of the database and the structure input. Here, the decomposition of tokens into words, definitions, NL-SfB classes and specifications will be treated. There will also be explained what formal rules there apply to create, read, update and revise (CRUD) words in the database without the use of the 'Database Manager KOB'. This implies how that database is functioning and interacting with the program, with the actual code. This process happens under the hat. How the 'Database Manager KOB' can be accessed and used by its users is treated within this chapter.

The program reads words from the database as structured by tokens. These tokens are built from words, definitions, classes, and specifications. From here on, we assume that a word in its enriched state is referred to as a token. Within the database, we can tokenize words by enriching them. This enrichment process captures dependencies between words and the system. The enrichment process provides the words as obtained from the requirements with knowledge and information. This structure provides the users to tokenize a certain word according to a certain enrichment principle. Figure 24 summarizes the structure of a token and its possibility to be enriched within the systems database by means of definitions, classes, and specifications.



Figure 24: Tokenization of word, definition(s), class(es) and specification(s).

The Tokenization of words functions as the fundamental data structure of the content within the database. The 'Database Manager BOK' is introduced within the system to accommodate all formal notations in a representative and concise way. The graphical user interface of the 'Database Manager BOK' is designed as follows:

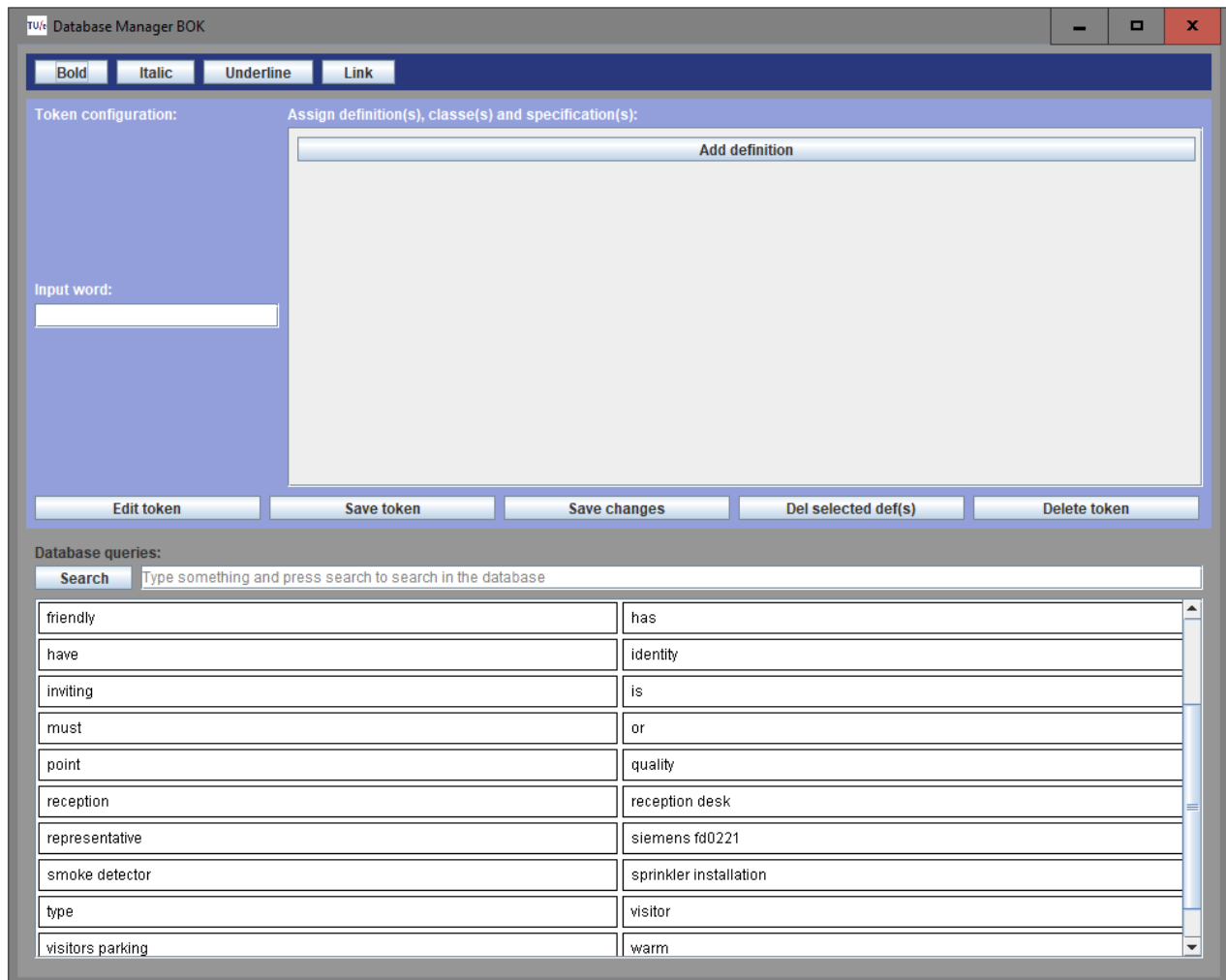


Figure 25: The graphical user interface of the 'Database Manager BOK' (Use case 2).

Words are basically the origin where knowledge and information will be attached to. This process starts by defining the words. Here, there is assumed that a word can possibly contain a fundamental linguistic definition, and definitions in terms of domain language. Here, domain language will be expressed by means of legal, geometrical, structural, building physical, material technical, financial, and aesthetic definition(s) (Niemeijer, 2011). These definitions that define the word(s) can be structured as follows:

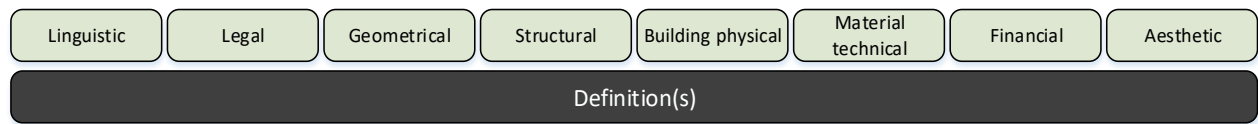


Figure 26: The variety of definition domains for word(s) enrichment.

Whenever words are defined, its allocation towards a system class needs to be assigned. Here, there is assumed to plug in the NL-SfB system to classify the System Breakdown Structure by means of a formal way. The essence of this process is to link a word where it applies to on a subsystem level. This helps in the following process to specify a definition of that particular word in relation to its allocation. This will be demonstrated within section 7. The words can be classified and allocated by means of the following structure:

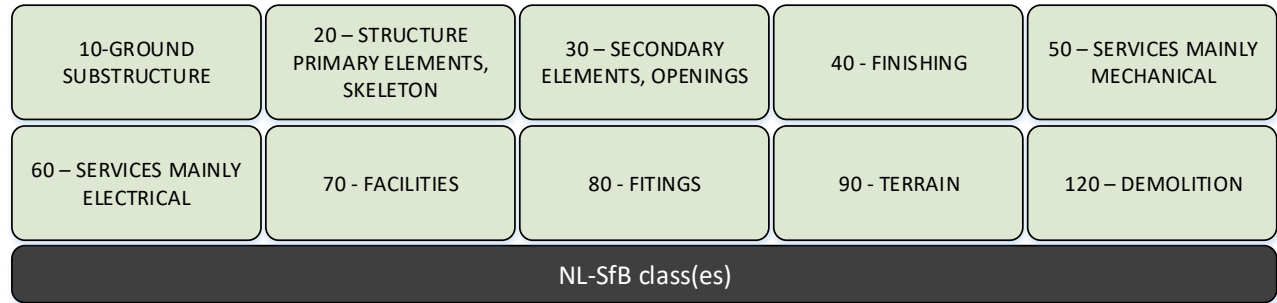


Figure 27: The variety of predefined classes as a function of the SBS.

The token that consists of words, as defined and allocated as a function of a class in relation with the system, can then be specified. This specification procedures is based on numerical and non-numerical attributes that describe a certain specification.

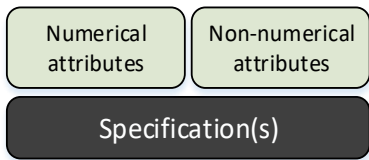


Figure 28: The structure of a specification.

From here on, the structure of an *enriched token* after the *enrichment procedures* within the database can be expressed by the following (fundamental) example:

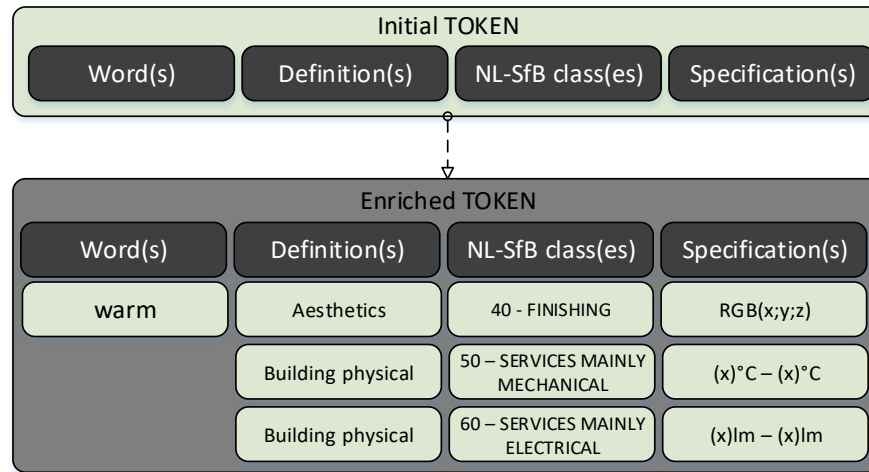


Figure 29: Fundamental structure of an enriched token.

The database is structured in a CSV file and consists basically of cells that relate by means of rows and columns. Here, every row within the CSV file defines a token. This will be elaborated upon the following sections. The data structure within the CVS file can fundamentally be illustrated by means of figure 30.

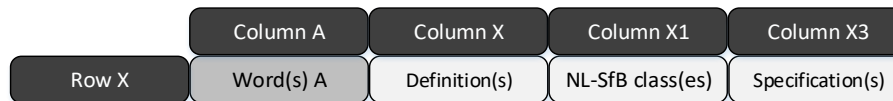


Figure 30: Data structure within the database.

Figure 30 defines the formal language in which information and data needs to be captured within the system its database. To allocate a word to a specific class within the SBS, the user commands “*<type>”, where <type> is one of the predefined classes from the SBS. Default is ‘other’ in this case, which is used to define the words its initial lexical definition. To specify a specification to an allocated word, the user needs to command “**<specification>”. This leads to the following representation:

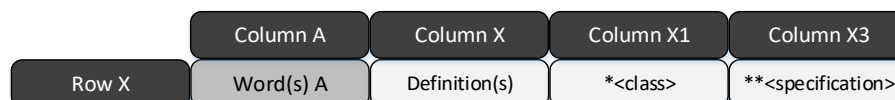


Figure 31: Formal representation of a token within the database.

The system is basically programmed to find the word in the first column, column A. Each cell that is positioned in the row of the word, row 1 in this case, will be assumed to be a definition cell. In other words, all cells after cell A1 will be assumed to be definitions of the word(s). This assumption stops until the program parses “*<type>”. This prompts the program to parse the input within the successive cells as a class that will be related to the SBS. All cells after “*<type>”, D1 for example, will be assumed to be a definition of the type as stated in cell C1. This process stops until the program reads in on “**<specification>”. Here, the program will be prompted to read a certain specification at the position of cell E1. All cells after cell E1, F1 for example, will then be assumed to be an additional definition on E1

The words within the tokens can also be enriched not by HTML links instead of textual descriptions on their definitions. The user can simply combine a textual description with a HTML by using the following: source , where ‘xyz’ is the link and ‘source’ is the linguistic description of the link. Figure 32 represents this format where a words, classes, and specifications can be enriched with a HTML link.

	Column A	Column X	Column X2	Column X3	Column X4	Column X5
Row X	Word(s)	 source 	*<class>	 source 	**<specification>	 source

Figure 32: Formal representation of a token enriched with HTML web links.

This page is intentionally left blank

7 Procedures for system use

7.1 Procedure for the translation of a mono-disciplinary requirement, Use case 1: TRANSLATE.

Within this paragraph, the operational functionality by use of the BOK system for the translation of a simple, monodisciplinary requirement will be demonstrated. This procedure is reported as a function of the early design process. The exact process of the use case of the tool is described and captured within section 5.2.1.

This paragraph demonstrates and concludes that the user(s) of the 'Bank of Knowledge' system are only required to fill in a certain 'mono disciplinary client requirement' within the input field, and to press the generate button in order to receive a product specification. The practical process will be demonstrated according to the following steps:

1) The mono-disciplinary requirement as obtained from the client's brief, *"The type of smoke detector is Siemens FD0221 or equivalent"*, needs to be extracted and imported in the input field;

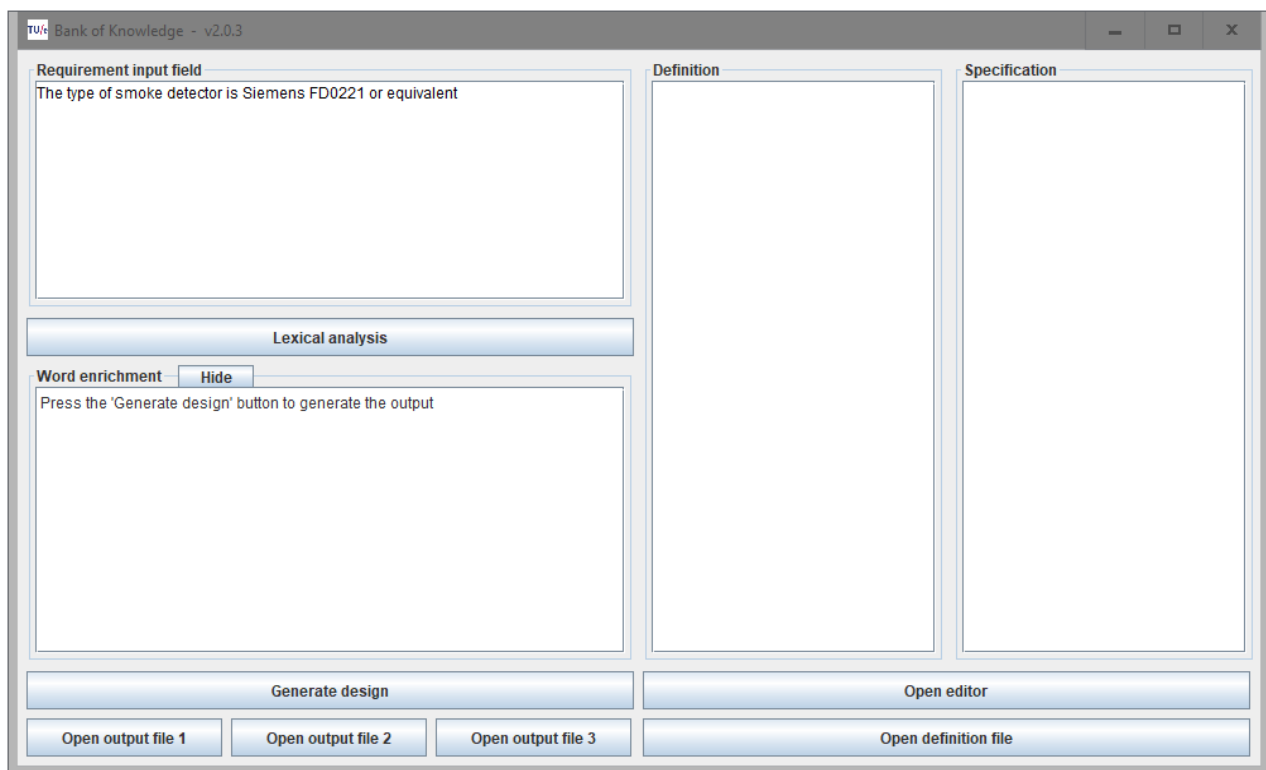


Figure 33: Input of a mono-disciplinary requirement within the 'requirement input field'

2) Whenever the syntax of the requirement is defined in terms of its linguistic composition, then the words as previously enriched in previous projects can be observed as follows:

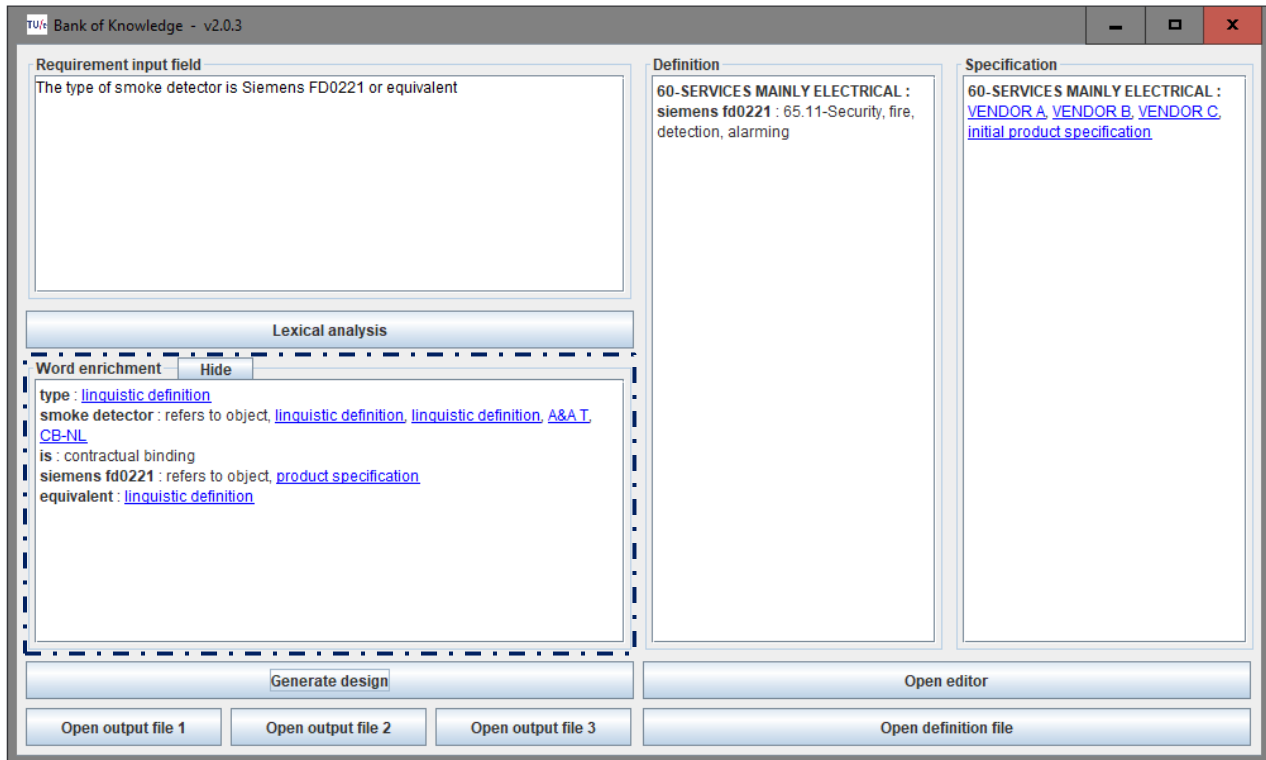


Figure 34: The enriched representation of the obtained words within the 'word enrichment' field.

This panel can be consulted by the user whenever the fundamental definitions of the words, as obtained from the requirements, are needed for certain purposes. This panel can be simply be covered by pressing the 'hide' command. This shall be the case for the next steps, here the 'Word enrichment' panel shall be covered.

The relevant words as determined by the database developers according to the principle as stated within 5.2.1, have been enriched by valid knowledge. The characteristics of the enrichment can be read by the description as stated in [blue](#) behind the word. The user can assume that these [characteristics](#) are referring to the links that were consulted for knowledge or information during the enrichment procedure of the word in previous projects. These are captured within the definition file which is the database. The database can be reached by pressing 'open definition file', this button is integrated to make the Database Manager BOK assessable from this application. This will be discussed later on in this report.

Research

Research Home ▶ Tools ▶ Art & Architecture Thesaurus ▶ Search Results

Art & Architecture Thesaurus® Online
Search Results

[New Search](#) [Previous Page](#) [Help](#)


Find Name: **smoke detector**



Logic:

Note: 2 results

[View Selected Records](#) [Select All Records](#) [Clear All](#)

First Previous Next Last
Page: 1

Click the  icon to view the hierarchy.
Check boxes to view multiple records at once.

- ☐  **photoelectric smoke detectors**
(smoke detectors, fire alarm system components, ... Components (hierarchy name)) [300052775]
detectors, photoelectric smoke
photoelectric smoke detector
smoke detectors, photoelectric
- ☐  **smoke detectors**
(fire alarm system components, protection system components, ... Components (hierarchy name)) [300052768]
detectors, smoke
smoke detector

[New Search](#) First Previous Next Last
Page: 1

[Back to top](#)

[Printer-friendly version](#)


 The J. Paul Getty Trust
© 2004 J. Paul Getty Trust
[Terms of Use](#) / [Privacy Policy](#) / [Contact Us](#)

Figure 35: The definition of the word ‘smoke detector’, within the ‘word enrichment field’, is defined by means of a Web Based thesaurus (Art & Architecture Thesaurus, 2017). This knowledge is captured within the system its database.

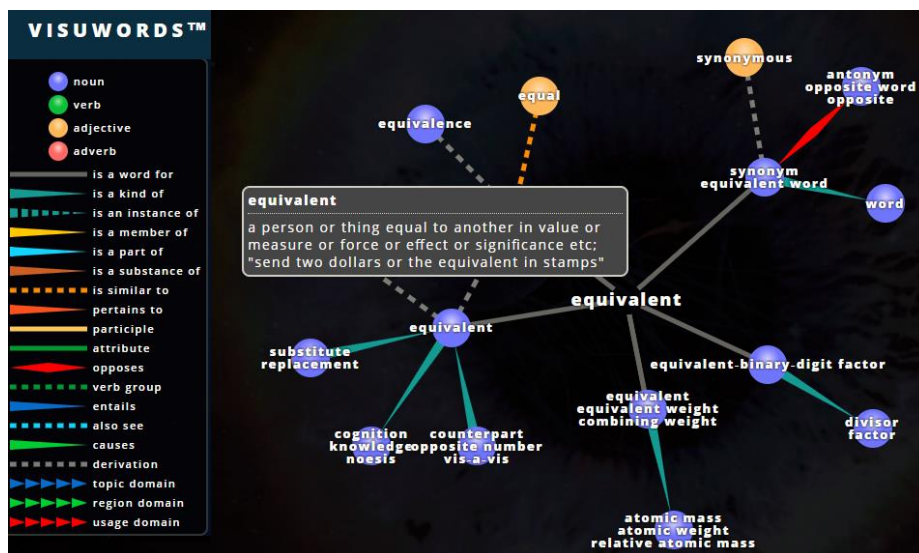


Figure 36: The definition of the word ‘equivalent’, within the ‘word enrichment field’, is defined by means of a Web Based thesaurus (Visuwords WordNet, Princeton 2017). This knowledge is captured within the system its database.

3) Within this step, the allocation of certain words in relation to an object on a subsystem level can be observed. This means that this word, as obtained from the requirement, has been allocated to this specific part of the system according to a certain system classification principle (NL-SfB in this case).

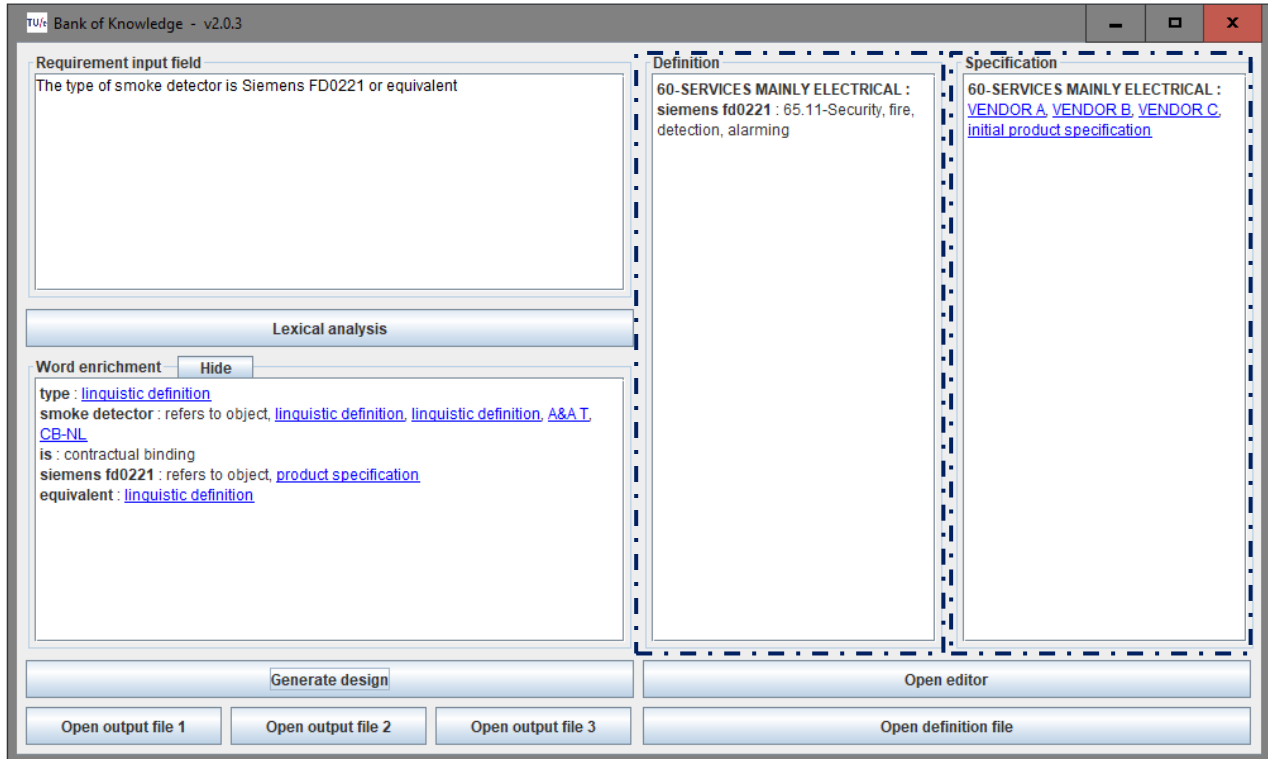


Figure 37: The allocation of the word 'Siemens fd0221' and 'equivalent' on a subsystem level within the SBS according to the NL-SfB as presented within the 'word to object allocation' field. Here, both 'Siemens fd0221' and 'equivalent' are allocated to the NL-SfB class no. 60. The specification of the 'Siemens fd0221' is expressed by the HTML link named as 'initial product specification', and 'equivalent' is expressed in specifications as obtained from 'Vendor A' – 'Vendor B' – and 'Vendor C' that deliver product with the same specification or better.

FDO221

Smoke detector, FDnet wide spectrum DA (C-LINE) (colored)

Building Technologies
 Contact
 Downloads
 A-Z Index

Site Explorer

[Building Technologies](#)
[Products](#)
[Fire Safety \(EN\)](#)
[Fire detection](#)
[Sinteso](#)
[FDO221](#)

FDO221

Smoke detector, FDnet wide spectrum DA (C-LINE) (colored)

Stock no. A5Q00016440
 Stock no. A5Q00016440

For the early detection of smoke-generating, flaming and smoldering fires.

Works according to the scattered-light principle with one sensor, optical forward scattering. Opto-electronic sampling chamber blocks disturbing extraneous light but optimally detects smoke particles. Signal processing with detection algorithms (DA).

The colored base must be ordered separately.

Attribute	Value
Standard	EN54-7, EN54-17
Protection category	with base IP43, with base attachment IP44
Operating voltage	12... 33 VDC
Operating temperature	-10... +60 °C
Storage temperature	-30... +75 °C
Quiescent current	180... 230 µA
Ext. alarm indicator	2 without sounder base 1 with sounder base
System compatibility	FDnet -> FS20, AlgoRex, SIGMASYS
Communication protocol	FDnet
Relative humidity	≤95 %
Dimensions (Ø x H)	100 x 46 mm
Color	according to customer requirement
Accessories	FDBZ293 Detector locking device, FDZ291 Detector dust cap

[For more information](#)

Text Size

Share this Page:

Download center

Get data sheets, manuals, brochures and more at our download center.

[Downloads](#)

For more information

[HIT online product catalog](#)
[I-Mall webshop](#)

Datasheet

[009409](#)

109

7.2 Procedure for the translation of a non-functional requirement, Use case 1: TRANSLATE.

Within this paragraph, the operational functionality by use of the Bank of Knowledge for the translation of a somehow fuzzy, non-functional requirement will be demonstrated. The exact process of the use case of the tool is described and captured within section 5.2.2. Here, the practical process will be demonstrated according to the following steps:

1) The non-functional requirement as obtained from the client's brief, *"The reception desk is the first point of contact for a visitor with the EEE. The reception desk has to be representative of the EEE quality and identity. Applicable key word are: representative, inviting, warm and friendly"*, needs to be extracted and imported in the input field;

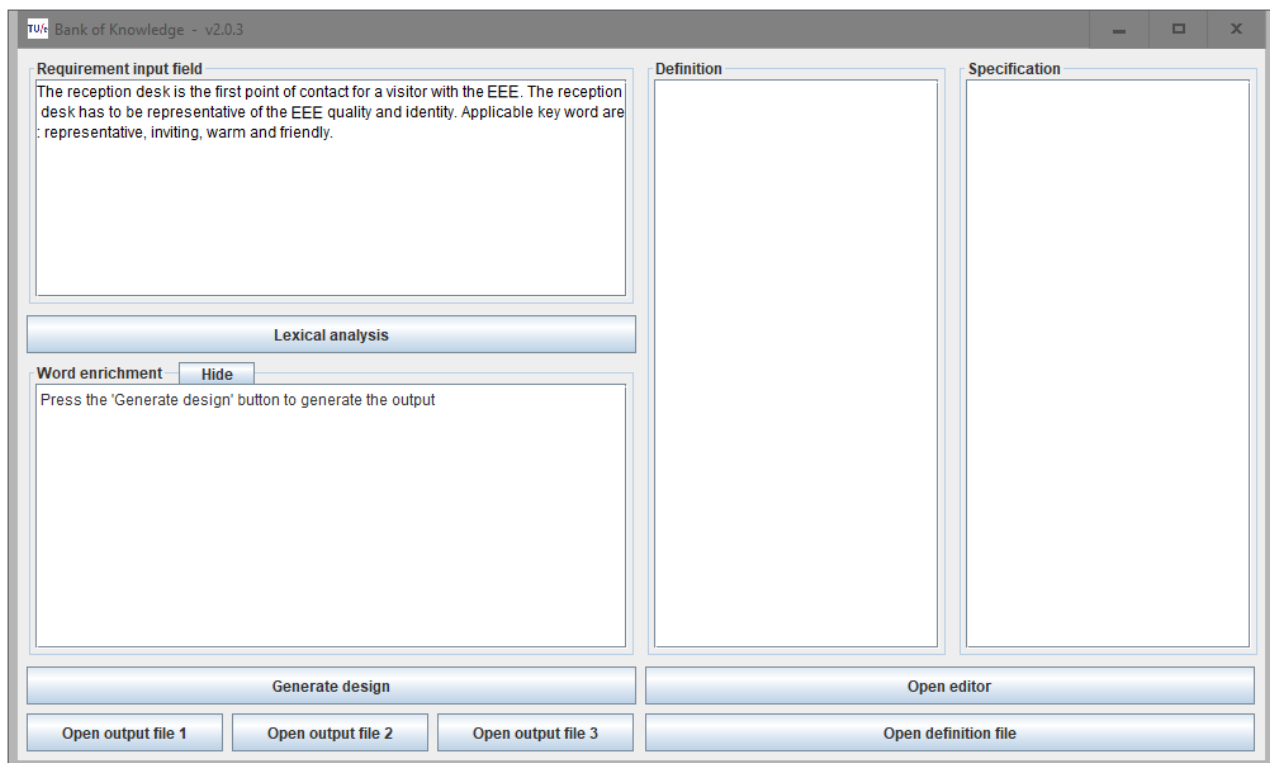


Figure 39: Input of a non-functional, somehow fuzzy, requirement within the 'requirement input field'.

2) Whenever the requirement is positioned in the input field, the automated ‘Lexical analysis’ can be executed. The program will guide its user automatically to the web based automated lexical analysis tool as developed by the Noah’s Ark group (2017).

3) Whenever the syntax of the requirement is defined in terms of linguistic composition, then relation between words from the sentence and the words that are stored within the systems database can be observed as follow:

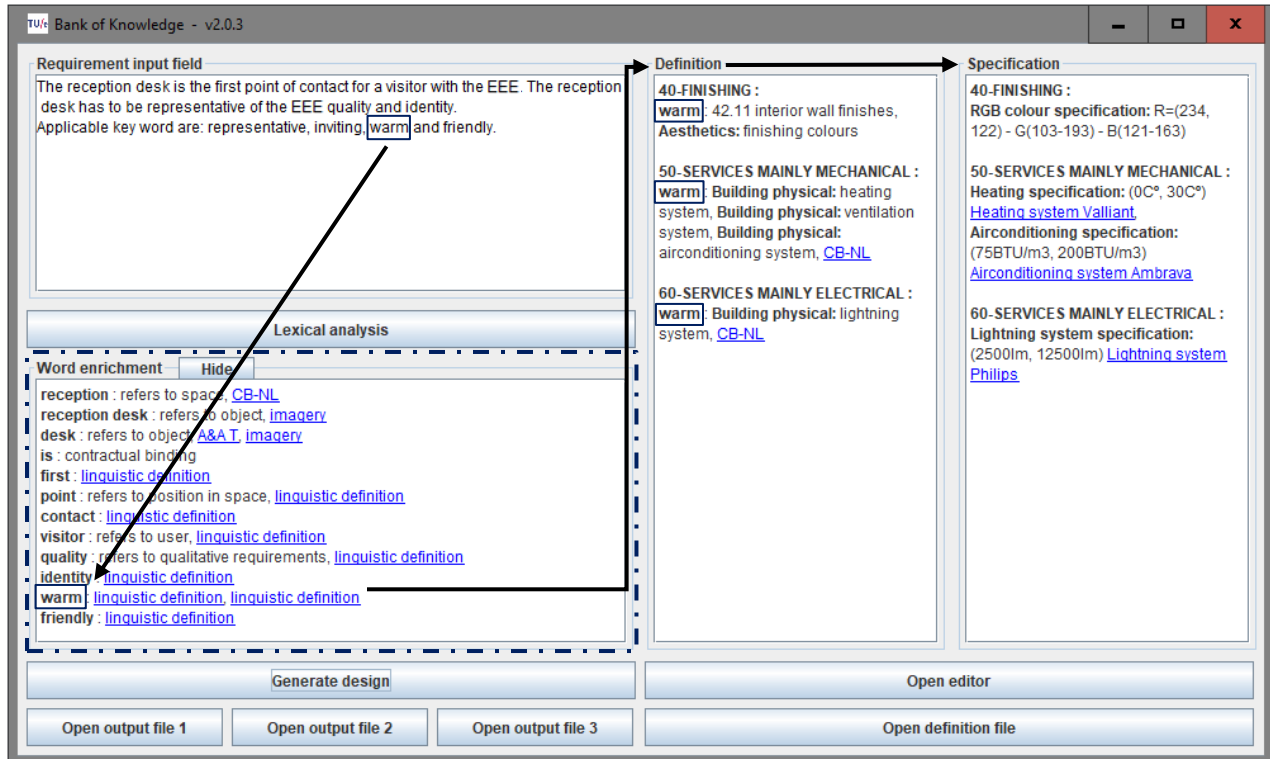


Figure 40: The enriched representation of the obtained words within the ‘word enrichment’ field.

The most important words within the requirement, as determined by the database developers, have been enriched by ‘valid state of the art’ knowledge. For the sake of brevity, there has been chosen to treat the enrichment, translation, allocation and specification path of the *keyword* ‘warm’ as a function of system design. In reality, each of the single words as stated within the input panel can be followed by the same path. The characteristics of the enrichment can be observed by the description that it contains behind the initial word as stated in [blue](#). The user can assume that these [characteristics](#) are referring to the links that were consulted for knowledge or information during the enrichment procedure of the word in previous projects. These can now be consulted to perceive a better understanding of the sentence and its meaning.

4) Within this step, the allocation of the words where the requirement is built from will be related to objects on a subsystem level. More specific, words with an ambiguous nature within the context of the requirement have been assigned to objects within the SBS.

Here, for instance, warm has been assigned in terms of ‘architectural finishes’. This could also link the word to an acting discipline. This link implies measures in term architectural specifications. Warm is also defined in terms of disciplines that contribute to the design of ‘heating’ or ‘electrical’ subsystems. The configuration of a space, where this requirement is referring to, is demarcated by those particular objects that define the performance of this space. This implies that the words need to be defined in terms of this demarcation in regards to disciplinary measures.

The accumulation of these mono-disciplinary measures, as derived from the words within the initial context of the non-functional requirements, form the configuration of the requirement. These mono-disciplinary requirements can be allocated to specific parts of the space. In this example, a mono-disciplinary requirement has been derived in terms of the predefined definitions such as ‘Aesthetics’ and ‘Building physics’ which will be treated within the next step.

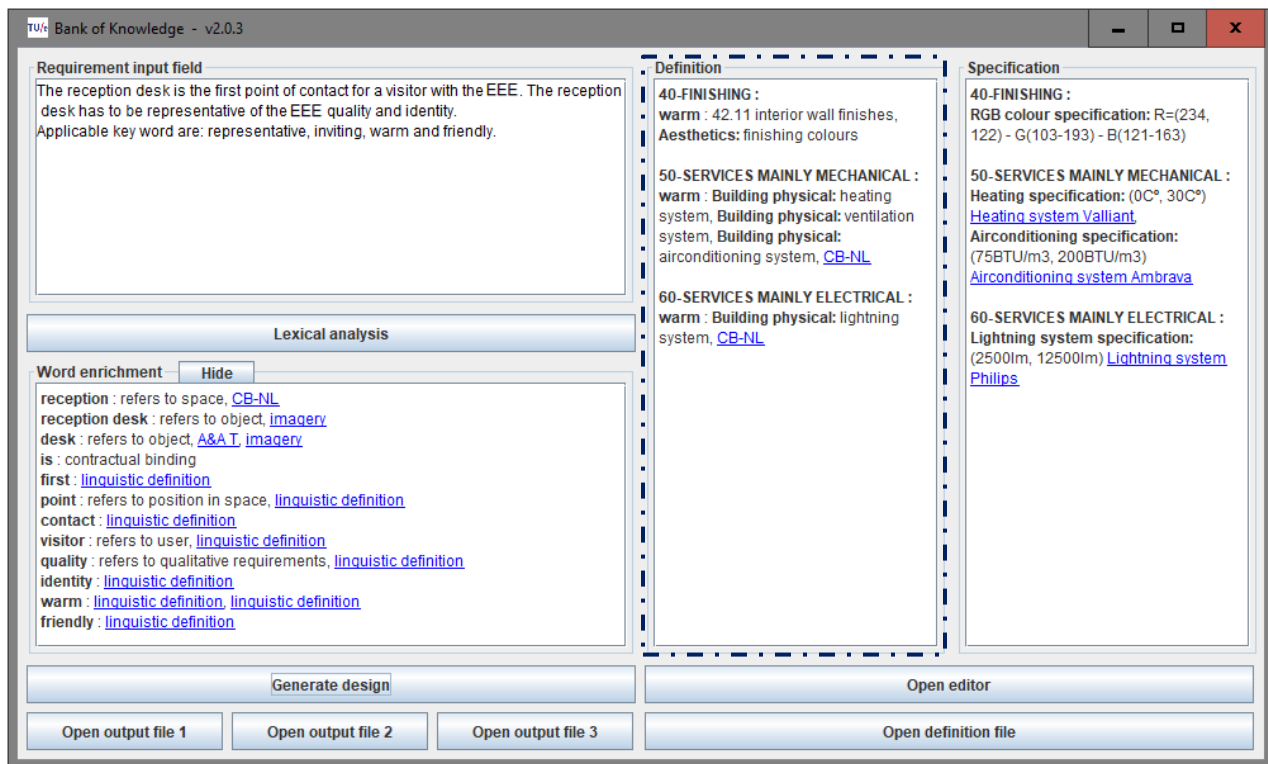


Figure 41: The allocation of the word ‘warm’ as obtained from the non-functional requirement on a subsystem level. In this example, ‘warm’ has been automatically allocated to ‘40-Finishing’, 50-Services mainly mechanical, and 60-services mainly electrical within the SBS within the ‘NL-SfB classification’ field.

5) Within this requirement, the design solution is not stated within the sentence itself. Therefore, the specification are simply obtained by automatically consulting the systems database for valid product specifications. These specifications communicate the performances of the word warm in relation to the parts of the system that is has been assigned to. These product specification are allocated and specified within the ‘word to object allocation specification’ column at the right hand side.

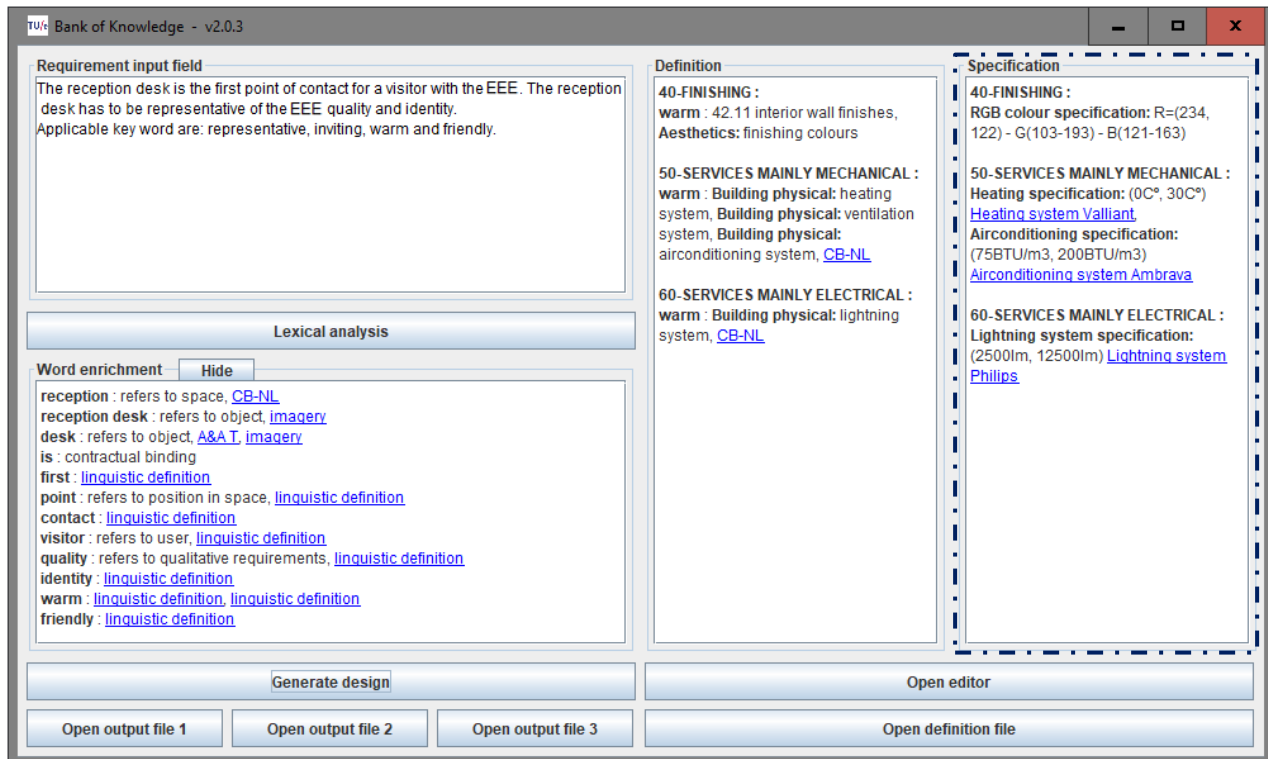


Figure 42: The valid product specification that communicate the product performance’s obtained systems database which are provided within the ‘word to object allocation specification’.

These mono-disciplinary requirements are distilled from the initial non-functional requirement as stated within the ‘requirement input field’ at the beginning of this paragraph. This process took a somehow fuzzy requirement to an unambiguous description that could be specified in terms of engineering information.

By this demonstration we can conclude that the definition of the applicable keyword ‘warm’ is translated into required ‘product specifications’ according to the methodology as obtained from this research that is accommodated within the ‘Bank of Knowledge’ systems operational functionality.

7.3 Procedure for database management, Use case 2: Database Management.

This chapter treats the second Use case of the system in which the CRUD principle will be discussed. This principle accommodated within the 'Database Manager BOK' which is designed and programmed to create, read, update and delete the system its database off concepts. In this database, all tokens are stored according to the principle as described within section 6.1.7. This database manager is designed with the use of layman in mind, and is therefore very user friendly. Within this paragraph there will be introduced how this part of the system can be consulted for database related activities. The first part of this paragraph consists of a brief introduction on the GUI of the 'Database Manager BOK'. This is illustrated within fig 43. beneath:

The screenshot displays the 'Database Manager BOK' application window. At the top, there's a title bar with 'TU/e Database Manager BOK' and standard window controls. Below the title bar is a menu bar with 'Bold', 'Italic', 'Underline', and 'Link' options. The main interface is divided into two main sections. The top section is titled 'Token configuration:' and contains a sub-section 'Assign definition(s), classe(s) and specification(s):'. This sub-section has a large text area with an 'Add definition' button. Below this is an 'Input word:' label and a text input field. At the bottom of the token configuration section are five buttons: 'Edit token', 'Save token', 'Save changes', 'Del selected def(s)', and 'Delete token'. The bottom section is titled 'Database queries:' and features a 'Search' button and a text input field with the placeholder 'Type something and press search to search in the database'. Below the search field is a table with two columns. The table contains the following data:

friendly	has
have	identity
inviting	is
must	or
point	quality
reception	reception desk
representative	siemens fd0221
smoke detector	sprinkler installation
type	visitor
visitors parking	warm

Figure 43: The descriptions of the operational functionality that are accommodated within the graphical user interface of the 'Database Manager BOK' (Use case 2).

1) The first step is to basically add a new token that will be captured within the system its database. Please note that the token within this example is basically 'imaginary' and introduced as a demonstration tool rather than a valid outcome. The system its database is structured according to the data format as presented within section 6.1.7. This implies that the database manager will prompt its user with descriptions on the composition of the token in terms of word(s), definitions, class(es), and specification(s). Within this step, we are simply going to configure the token 'cold'. This is done by means of the following procedure.

1.1) The token 'cold' has been defined as a word within the 'input word' field. The word 'cold' has been 'defined' by means of a 'linguistic' definition that is described as a 'certain temperature which is < 19C °

1.2) In this situation, the token 'cold' has been assigned as a word 'cold' that is defined in terms of its linguistic definition. In de upcoming steps will demonstrate how a definition can be assigned to a certain word.

The screenshot shows the 'Database Manager BOK' window. The 'Token configuration' section on the left has an 'Input word:' field containing 'cold'. The 'Assign definition(s), classe(s) and specification(s):' section on the right has a 'DEFINITION' dropdown menu open, showing a list of categories: 10-GROUND, SUBSTRUCTURE; 20-STRUCTURE PRIMARY ELEMENTS, SKI; 30-SECONDARY ELEMENTS, OPENINGS; 40-FINISHING; 50-SERVICES MAINLY MECHANICAL; 60-SERVICES MAINLY ELECTRICAL; 70-FACILITIES. The 'NONE' dropdown is also open, showing a list of classes: linguistic, legal, geometrical, structural, building physical, material technical, financial. The 'specification(s)' field contains the text 'certain temperature which is < 19C°'. At the bottom, there are buttons for 'Edit token', 'Save token', 'Save changes', 'Del selected def(s)', and 'Delete token'. Below the configuration section is a 'Database queries:' section with a search bar and a table of results.

Database queries:	
Search: Type something and press search to search in the database	
applicable	are
banaan	banana
cold	contact
desk	eee
equivalent	first
friendly	has
have	identity
inviting	is
must	or
point	quality

Figure 44: The fundamental 'linguistic definition' of the token 'cold'.

2) The second step within this procedure is to assign a 'NL-SfB class' to a token. This classification can also be enriched with a certain specific 'NL-SfB class definition'. In other words, the definition of 'cold' can differ from its initial definition whenever considering a class that this word applies to, this could declare ambiguity. This will be discussed later on within this chapter.

2.1) Adding a definition can simply be obtained by pressing the 'add definition' button. The word 'cold' has been assigned to the NL-SfB class '50-Services Mainly Mechanical' by means of a specific 'NL-SfB class definition' that is described as 'a certain cooling system'.

2.2) In this situation, the token has been assigned and defined to a specific NL-SfB class which allocates a token on an object level within the SBS. The definition of 'cold' has been translated to what 'cold' can mean in terms of the NL-SfB class '50-Services Mainly Mechanical'.

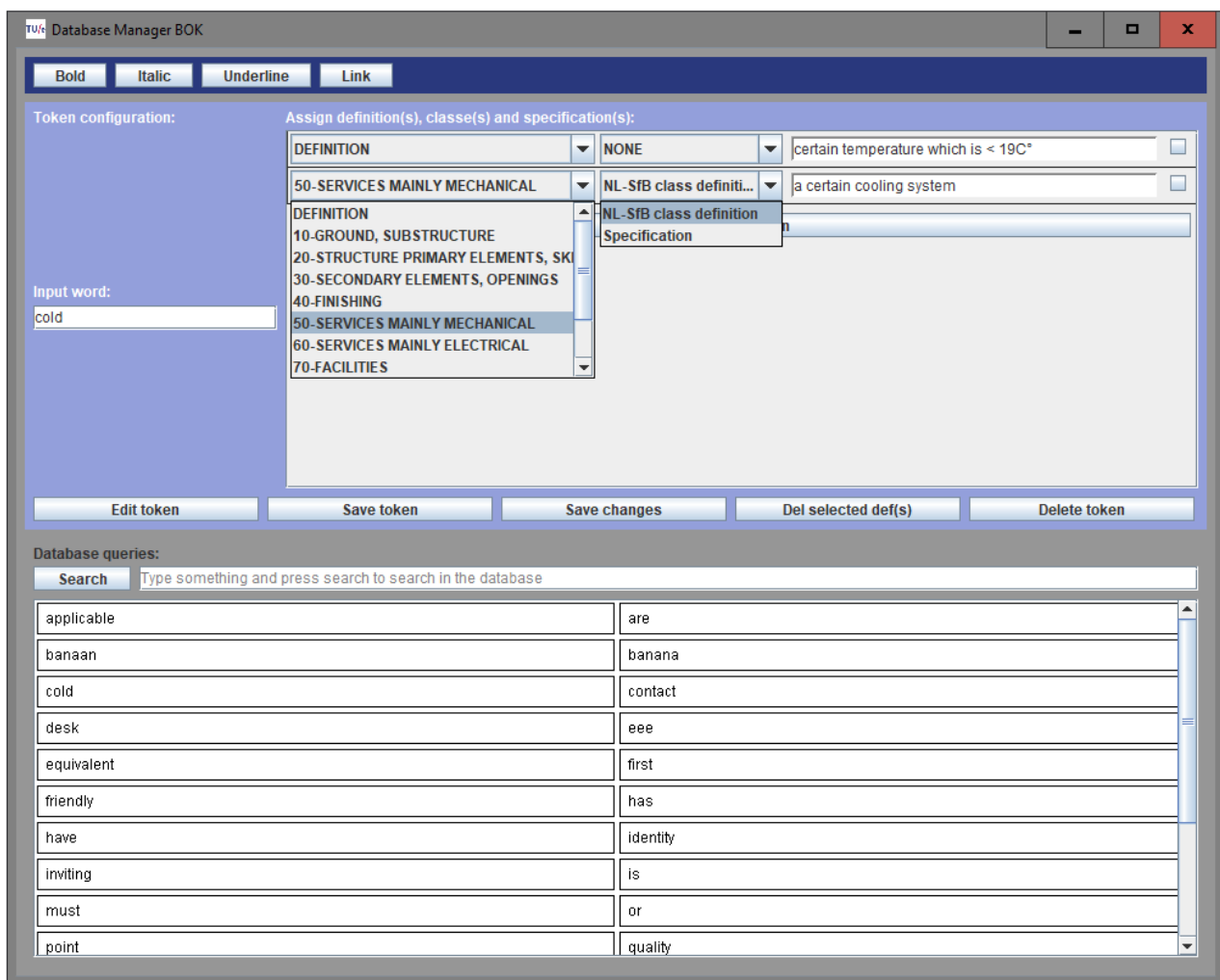


Figure 45: The allocation of the 'token' on a system level by means of a 'NL-SfB class' and its 'NL-SfB class definition'

3) The token 'cold' could also be defined, allocated and assigned to the same NL-SfB class; but with another application domain. Here, cold could have been defined as the amenity that users can have by entering a 'cold' room. Cold could both be defined in terms of 'Linguistics' 'Aesthetics' or 'Building Physics' and so on. This is mostly the case for ambiguous requirements which will be demonstrated later in this chapter.

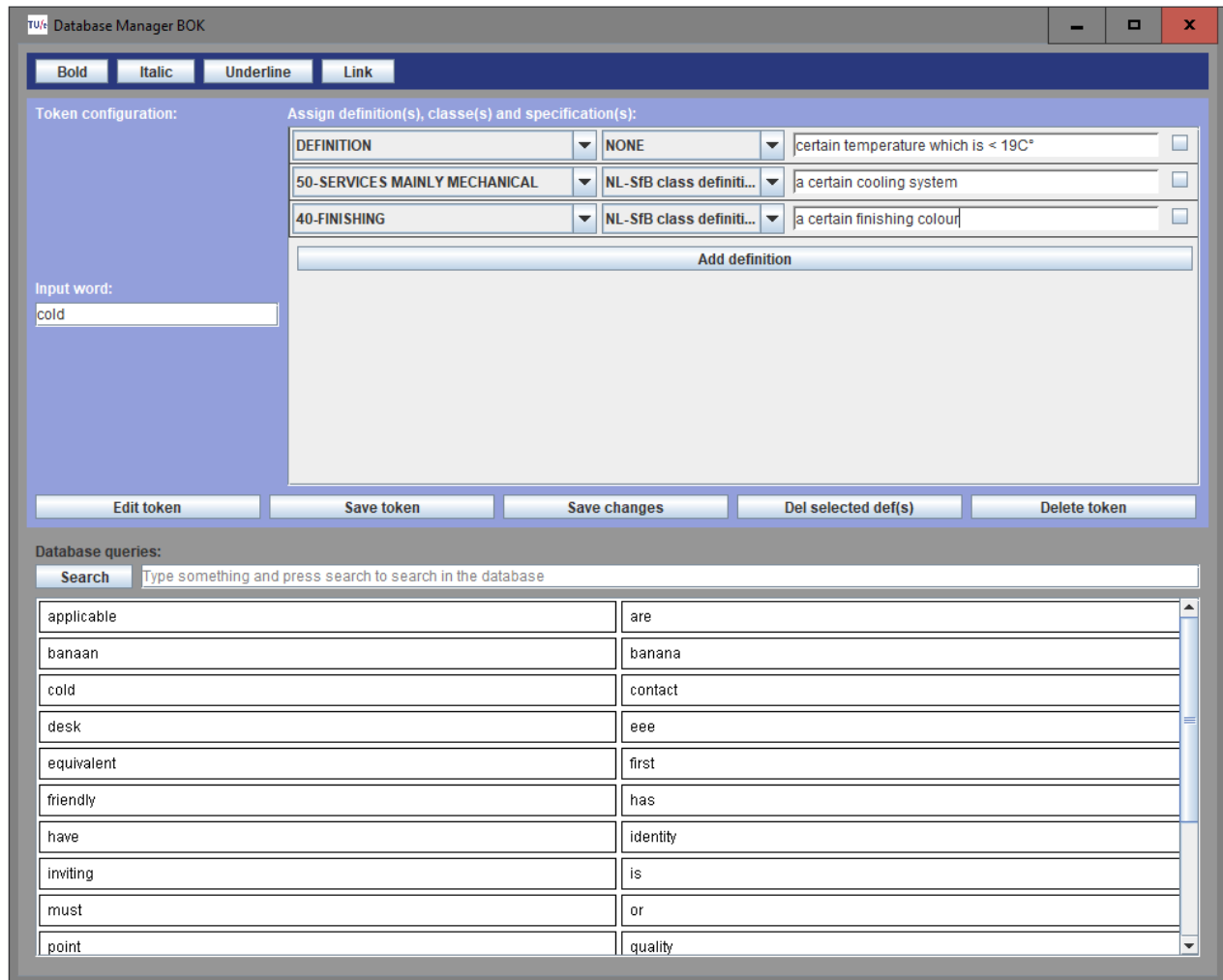


Figure 46: The allocation of the 'token', with the word 'cold', on a system level by means of a second 'NL-SfB class' and its 'NL-SfB class definition' do declare ambiguity regarding the definition and interpretation of the word 'cold'.

4) Given the ‘word’ its ‘definitions’ and its allocation on a object level, there can be specified and constraint how this token needs to perform. Capturing a certain product specification within the system its database is what aimed for in this step. As mentioned within step 1, the content of this token is far away from valid and is simply introduced as a tool for demonstration. In this step, the specific domain related definitions will be specified and constraint in (imaginary) product specifications.

Database Manager BOK

Bold

Italic

Underline

Link

Token configuration:

Input word:

cold

Assign definition(s), classe(s) and specification(s):

DEFINITION	NONE	certain temperature which is < 19C°	<input type="checkbox"/>
40-FINISHING	NL-SfB class definiti...	a certain finishing colour	<input type="checkbox"/>
40-FINISHING	Specification	cation: R=(234, 122) - G(103-193) - B(121-163)	<input type="checkbox"/>
50-SERVICES MAINLY MECHANICAL	NL-SfB class definiti...	a certain cooling system	<input type="checkbox"/>
50-SERVICES MAINLY MECHANICAL	Specification	itioning specification: (40BTU/m3, 150BTU/m3)	<input type="checkbox"/>

Add definition

Edit token

Save token

Save changes

Del selected def(s)

Delete token

Database queries:

Search

Type something and press search to search in the database

cold	contact
desk	eee
equivalent	first
friendly	has
have	identity
inviting	is
must	or
point	quality
reception	reception desk
representative	siemens fd0221

Figure 47: The ‘specification’ of the token in terms of ‘product performances’.

5) After the formalization of the 'token', the Bank of Knowledge will review this concept whenever it is commanded to translate this word. This results as follows within the BOK GUI:

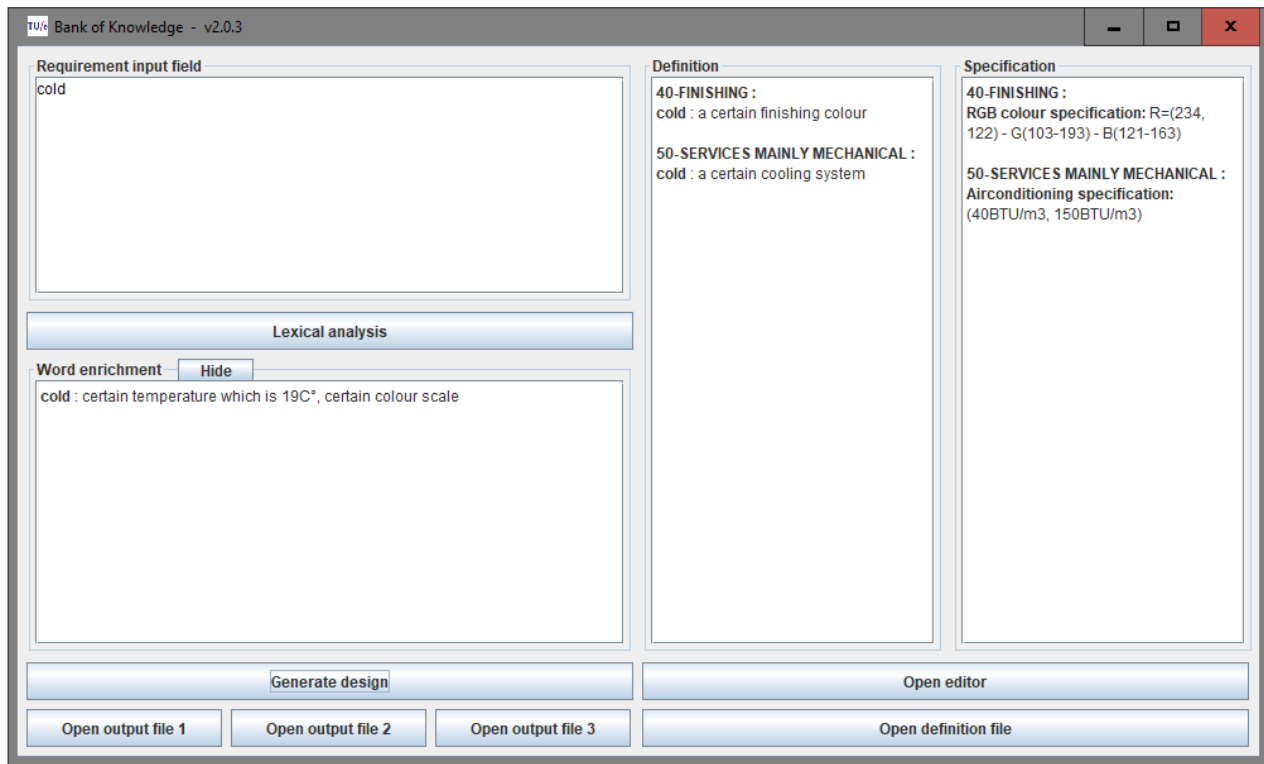


Figure 48: The translation of the token 'cold' in product specifications by means of the BOK system.

6) In this step, the example of 'the translation of a 'non-functional requirement' will be treated. Here, the definition of the word 'warm' within the initial requirement can instigate ambiguity by interpretation and translation.

Having a requirement that is mono-disciplinary from nature, a requirement that can be assigned and allocated to one party and part of the SBS, is desired by the developers of the database. This makes it relatively simple to define, allocate and assign a certain specification to a certain requirement. In cases of ambiguous words, or word combinations within a sentence, we need to derive and formulate a specification by means of a valid methodology or technique since this is unknown. This procedure is known to enrich the database with new concepts that are stored as tokens. These findings will translate the definitions of the words, in relation to its class, to technical specifications where valid design decisions can rely upon. Here, the following qualitative and quantitative methods and techniques can be introduced to derive (valid) specifications for 42.11 binnenwandafwerkingen; afwerkklagen (example paragraph 6.2):

- Literature;
- Reference projects;
- Questionnaires;
- Data analysis;

For the practice, and sake of brevity, we assume a (yet) practical data analysis to gain insight in the bandwidth of colours that the 42.11 binnenwandafwerkingen; afwerklagen should contain. For this case, the google.com search engine can be consulted to derive the colours that satisfy the finishing colours of the space in regards to the initial requirement. Here, we shall search on the following keywords: warm room colours given the fact that we have found a clear definition of the word warm, within the enrichment stage, in terms of architectural colour states.

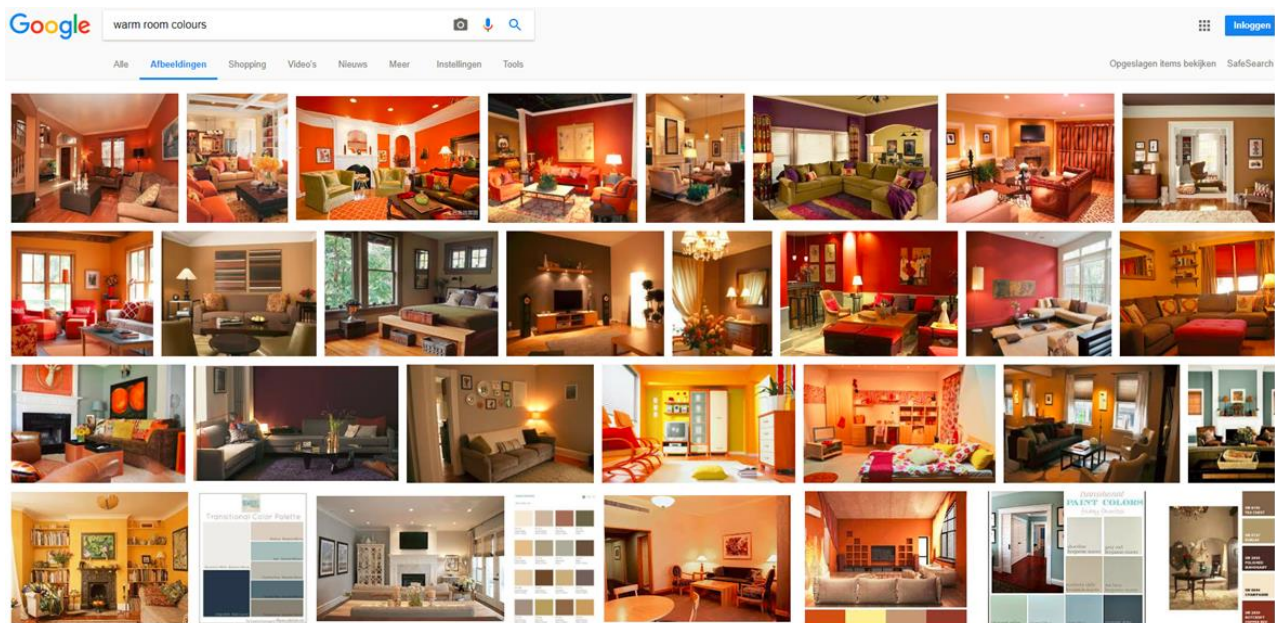
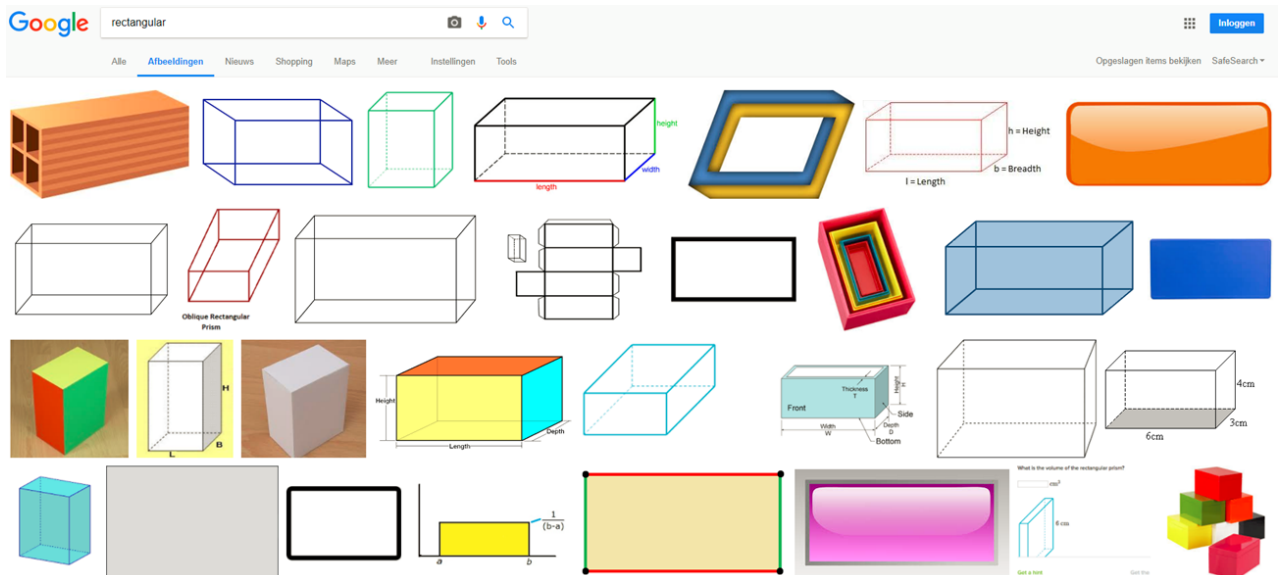
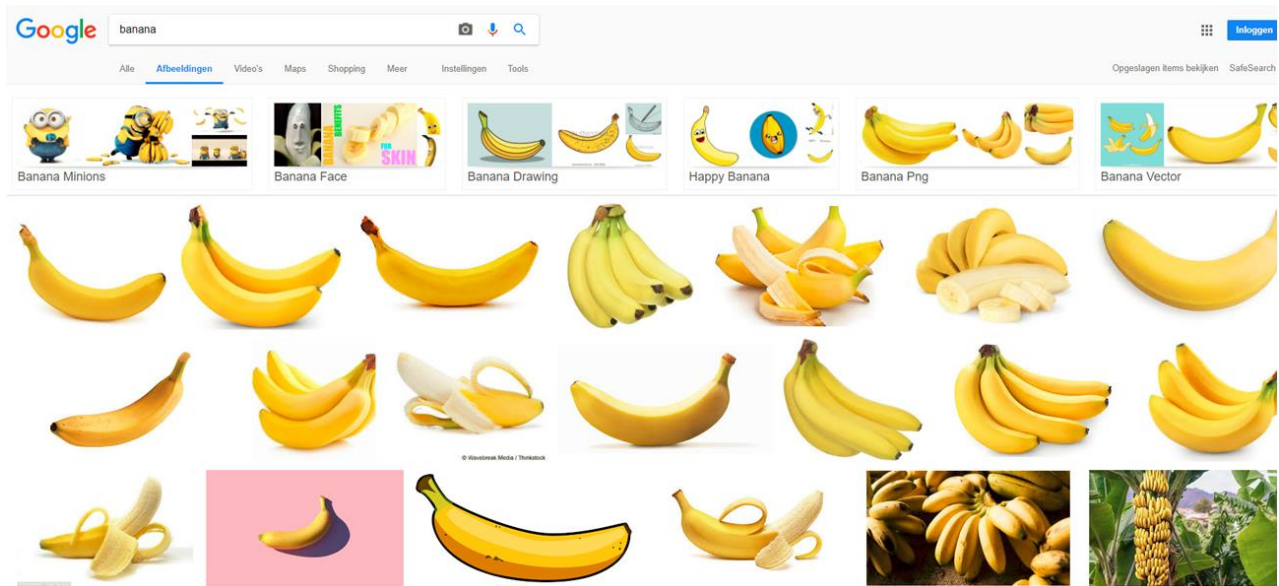


Figure 49: The findings of the 'google search engine' after entering the search keys 'warm room colors'.

But how valid are the searches from google where our data set will be gained from? How valid is the database that can be delivered. Let's test what the engine finds after searching on the keywords 'banana' and 'rectangular'.



Of course, this is no scientific approach. But this, yet very simple, technique could provide the possible ranges of colours that can be used by the determination of valid colour ranges that satisfy the requirement in terms of finishing colours. Whenever determined, then these colour ranges can be defined as a specification; a bandwidth in which design decisions can be taken in terms of architectural finishing colours. If a larger amount of pictures will be analysed, then we could derive statutes within the data that we can formulate as knowledge.

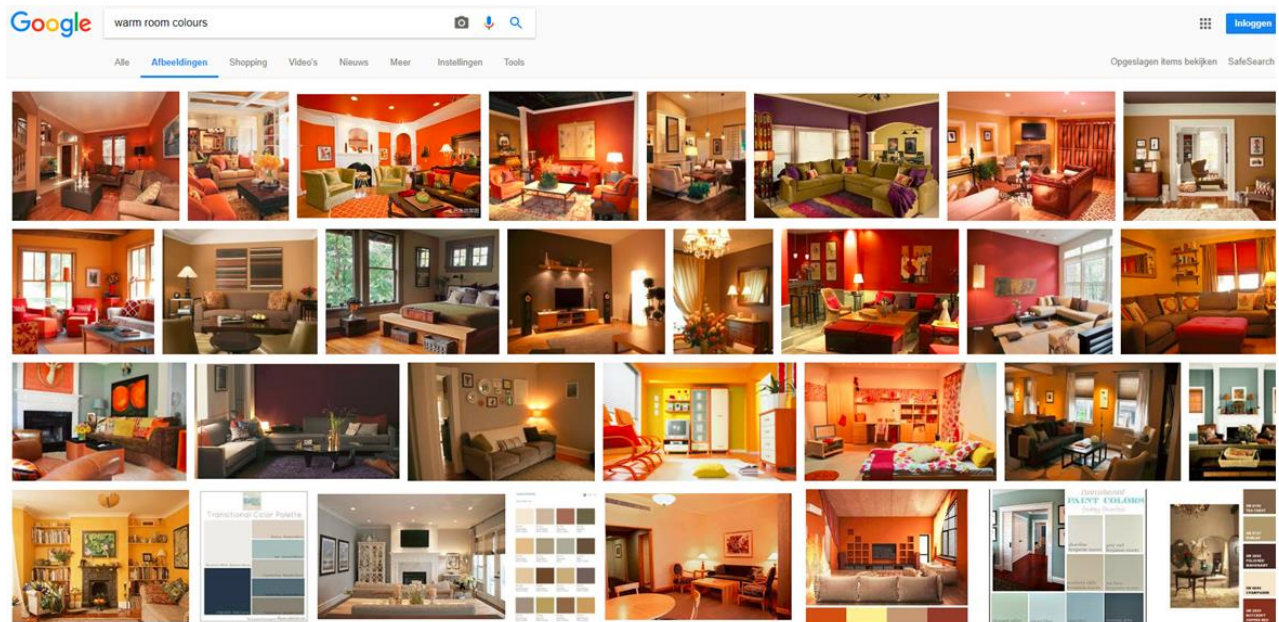


Figure 52: The findings of the 'google search engine' after entering the search keys 'warm room colors'.



Figure 53: Photo sample as obtained from the google search on 'warm room colors'.



Figure 54: Extracting a color sample of the obtained picture to capture knowledge on colors that have been defined by individuals on the globe as ‘warm room colors’.

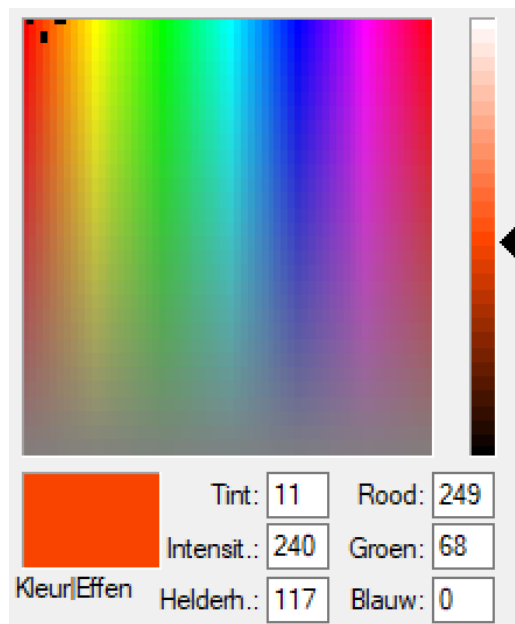


Figure 55: The color sample that defines ‘warm room colors’ in terms of TIH and RGB specifications. These specifications can be converted to any kind of color scales that product manufacturers use; such as the RAL scale.

We observe the outcome of the monster(s) by taking their TIH or RGB specification. Then, these can be translated formats such as, for instance, RAL specifications where vendors might specify their products with. Imagine how specific this could get if a large amount of pictures would be analyzed; it could be a valid given as stored within the KOB its database. This is basically what the system aims for: valid expert advice which grows as a function of time.

By this relatively simple and practical technique we could store knowledge that contains colors as a function of a certain lexical definitions. Then, from here on, this knowledge and information can be consulted each time within the tool by these type of specific queries for new projects.

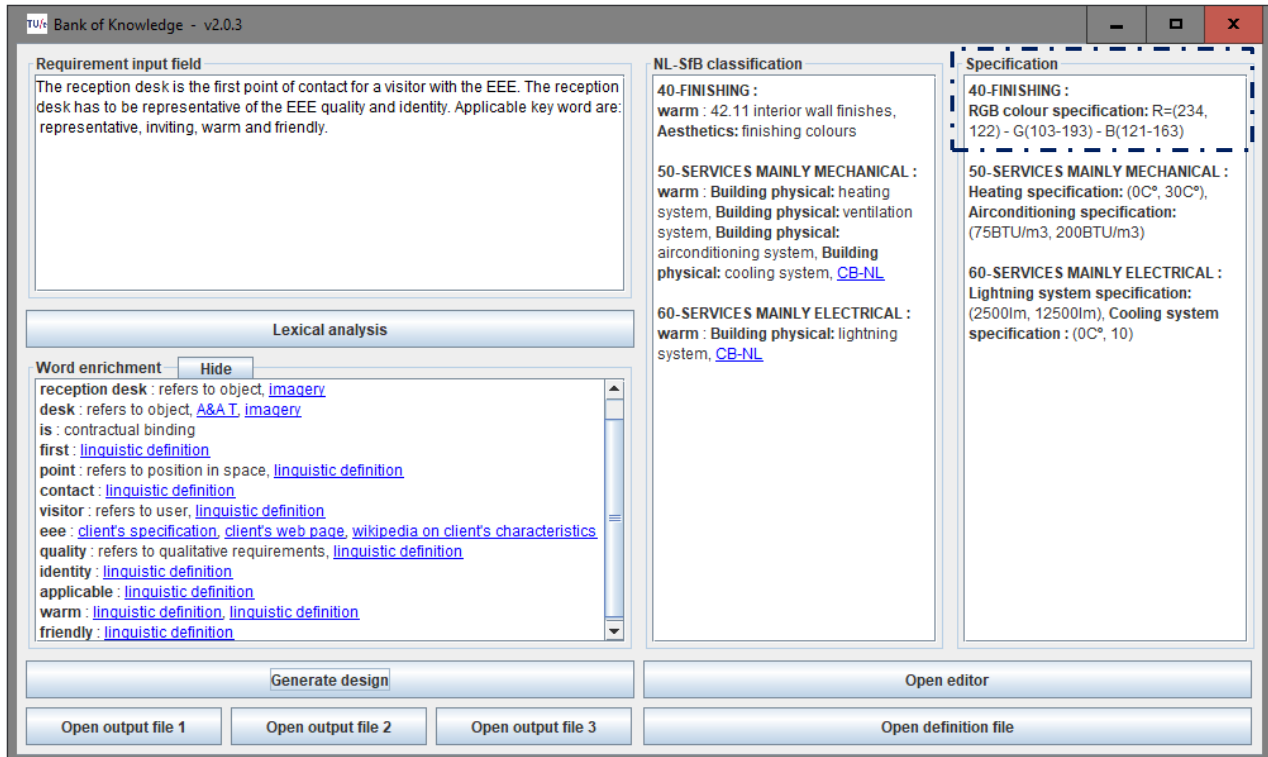


Figure 56: The final representation of the mono-disciplinary requirement, the specification, is stated within the ‘word to object allocation specification’ screen.

This mono-disciplinary requirement is distilled from the initial non-functional requirement as stated within the ‘requirement input field’. This process took a somehow fuzzy description to a more unambiguous description that could be specified. Here, the definition of the applicable keywords such as ‘representative, inviting, warm and friendly’ are translated into specifications according to this methodology as presented and accommodated within the tool its operational functionality. This mono-disciplinary requirement is distilled from the initial non-functional requirement as stated within the ‘requirement input field’. This process took a somehow fuzzy description to a more unambiguous description that could be specified.

By this demonstration, the definition of the applicable keyword 'warm' is translated into specifications according to the methodology as obtained from this research that is accommodated within the Bank of Knowledge its operational functionality.

This page is intentionally left blank

8 Conclusion

This chapter treats the conclusions for research question 1 to 6 and recaps by answering the main research question. These conclusions are made according to the findings from the review of literature, the interviews during the in-house practices, and the development process and form the conclusions of this research.

1) What client specification procedures are there in use within the design process, and how does Systems Engineering support these procedures?

The client specification procedures of the design phase and the systems engineering process have been identified by means of a literature review and the in-house practices. There is found that ambiguous requirements, as stated in the client's brief, evolve to unambiguous requirements as a function of the design process. This is the case assuming an iterative design process where there is a possibility to start dialogues with the client on the interpretation of requirements. Then, these requirements will evolve from ambiguous to unambiguous as a function of the design process assuming that they will be fine-tuned parallel to this iterative process. Capturing the correct interpretation and understanding of the client needs is mainly done iteratively during the design stage. The understanding of the set of requirements and the need of the client in an early phase is very crucial. This due to the fact that the decisions as taken in the early stages of the project are often the most crucial decisions as the impact of revisions later on in the project will have significant impacts on both finance and time management.

There is observed that the translation of client requirements are heavily relying on organizational experiences from the past. That is mainly the core where design decisions are relying upon. Yet, there is found that a lot of these organizational experiences from the past due to client and designer interaction, translation procedures, verification procedures and decision making are never captured and stored by means of a certain ontology, standard and semantic. Therefore, reusing this knowledge is almost impossible, it's just not there. Pragmatic knowledge as gathered by experiences of designers and engineers will be used in the heat of the moment to assess whether solutions may satisfy the client's requirements. Currently, at this point, a lot of human interpretations will come in to play which can easily cause errors due to misinterpretation. A significant improvement of implementation of systems engineering can be found in a more standardized way of working.

2) What variety of client requirement types are known within the design process, and which of these carry risk in terms of non-conformity?

There is researched upon how requirements are structured fundamentally. This provided the opportunity to identify what types of client requirements there currently exist within the AEC domain. There is found that requirements can be defined in two ways, either by their fundamental

linguistic structure or their classification towards prioritized needs. The structural composition of a certain requirement defines the initial step for methods to define requirements. The structure of a requirement can be captured with quantifiable, relational and qualitative descriptions. These define the initial structure of a certain requirement and where they are applying to. This structure can be expressed by a certain value, a relation or a qualitative description. These indicators reflect the level in which a requirement is measurable or the level of ambiguity that a requirement contains. These levels imply the complexity of translating a requirement into product specifications.

Requirements are also found to relate to a certain need. This implies that client needs can be expressed in terms of client specific requirements. This process, to absorb a certain need within a certain requirement, states the second step for methods to define proper requirements. The interpretation of the requirements obtained from the requirement analysis requires client validation. This validation procedure with the client is crucial to ensure that right interpretation is captured. This approval on 'design task' declares uncertainty for both the designer as client. Professionals within the Architecture, Engineering and Construction industry are often found to work according to traditional organizational ethics rather than by knowledge. This possibly implies the unconformity about a formal language among the parties, in which the system gets communicated captured.

The phenomenon of non-conformity regarding client requirements boil basically down to the structure of fundamental requirements. Both structure and composition of a requirement can affect human interpretation. Many misinterpretations of requirements are often occurring during early design stages. There are various layers in which ambiguity can occur. These can occur by writing, reading, and communicating requirements. There can be stated that requirements which have the biggest impact are the requirements which have the highest priority towards the client's needs. This also implies the level of risk that a requirement can contain.

3) What is the current practice in the AEC industry for translating client specific requirements into product specification, and how do verification procedures safeguard these?

The current practice in client requirements verification and validation, as often implemented by field experts on a more practical level, is visualized by a schematic representation as captured within chapter 4. The core steps of both the translation and verification procedure are captured within activity diagrams. There must be mentioned that the shift due to 'modern work ethics' within the AEC-domain due to integrated contracts require more than the processual knowledge an average verification manager. This relates more to the domain of Design Management which increased in complexity due to deformations within the construction industry. The verification procedures take place after a phase in the project is finished. This is a pragmatic process where the quality is heavily relying upon the interpretation of the verification manager. The systems engineering approach has heavily improved this procedure relative to the unstructured traditional

methods. However, there is found that verification managers are required to have a decent understanding of the Domain Specific Language in which systems are configured, captured and communicated. The SE approach contributed by means of its methodical approach.

4) What can automation, for translating client requirements into product specifications, contribute to the design process?

Automation by translating client requirements can be very beneficial. Especially for coping with a demand specification which specifies a complex project where both the designing parties as contractor are unfamiliar with. In these cases, when budgeted, extended requirement analysis can be introduced for implementation. This basically means that the standard requirement analysis process will be extended by means of time. However, the main findings why extended requirement analysis are not introduced that often can be explained by the following reason, given: the AEC sector is used to start designing right away and adjusts its design during the iterative design process. This causes the insufficient time that there is taken to correctly interpret the need of the client. Another reason can be explained by the fact that the investment costs of extended requirement analysis are earned back after a tender is won. In reality, not every tender is won which makes this an unprofitable strategy to introduce this means for each project.

Then one can state that it is very useful to have specialists reviewing the necessity of such extended requirement analysis since this can be very profitable for contracting complex projects, especially by collaborating with specific clients. However, even these specialists are not capturing their decisions by means of standards and semantics. If they would, then this could prevent a lot of rework by each project. This due to the fact that reasoning by decision making is captured by means of standardization that can be reused. This was also found as one of the main preconditions of automating the translation procedures.

The few time that there is budgeted for requirement analysis implies the need for a certain (automated) information system that can be consulted within the early design stages to consult for specific queries. These type of systems could provide knowledge on what decisions have been taken by the translation of certain requirements from previous projects. This can be very beneficial for saving budget for decision making during the early design stages. There are numerous buildings in the world; yet a slight of their design decisions captured which is a pity.

5) What are the current techniques within the AEC-domain, by means of automation, to translate product requirements into product specifications?

Currently, as discovered by the literature review and the in-house practice, there are no such information systems in which requirements can be fed to obtain product specifications. The domains of data and text mining are coping with the conversions of chunks of data and text into informative representations, but these are not implemented within the specific domain of AEC to

obtain this research initiative its objectives. Industries where data and natural language is functioning as the core of business are known to use a variety of data- text mining, natural language processing, and speech detection techniques to stimulate certain business processes. However, there are no information systems in use within the AEC domain, as found by this research, that can convert natural language to product specification. There are various relational databases and electric requirement models that can store requirements according to semantics to stimulate interoperability. However, these are stored without a unified formal language or semantics. The only standards that are mainly captured within these databases are organizational from nature. This makes interoperability difficult given the fact that ambiguousness can occur by collaborating with both clients and external parties.

The CBNL is currently contributing to formalize a formal language in which concepts that are in use within the AEC are defined and stored unambiguously. This could be used within the future development of such systems, yet the CBNL is still developing. There must be stated that the lack of such information systems is not due to the technology, as reviewed within the topics of the literature review, but rather to the missing formal language and semantics that are required to capture and process such information to knowledge. Thereby, numerous buildings have been manufactured for mankind, yet, no standards are applied for storing specific information regarding design decisions which can be used for the implementation of knowledge based systems. This is very useful information given the fact that design decisions as made in the past, could be used to predict the outcome of design decisions made for the future.

6) Is it possible to develop a method that translates and stores physical, functional, and non-functional requirements into product specifications by means of automation?

Rule checking is currently used for validation of quantifiable requirements in building information models. Numerous researchers have contributed to the development of automated rule checking techniques. This is currently the bleeding edge technology within the domain of Building Information Management. However, this research has found that the initial translation procedure, from requirement to product specification by means of automation, is not researched upon within the AEC. The interviewees emphasized that it seems a complex task to translated physical and functional requirements by means of automation, given the fact that information on design decisions from the past have never been captured by means of standards and semantics. There is no knowledge captured in a formal way that can be used to feed such systems. There is a lack of lawfulness for knowledge capturing within the AEC-domain. The decisions from the past are not usable given these circumstances. This makes it even more challenging to initiate attempts to automate this process for the case of non-functional requirements. The interviewees mentioned that it is very difficult to automate the translation procedure of non-quantifiable, more qualitative requirements given the previous reasoning. Non-quantifiable and qualitative requirements are known to have different meanings for different human beings. The interpretations might strongly vary among a random human population. There are no standards in what qualitative requirements

can be compared to. This could be done whenever knowledge as gained from previous projects is captured in an adequate way. A client could be very satisfied if a designer could show him what he designed according to a certain non-quantifiable requirement. This might improve the client / designer interaction.

Among the interviewees, having a system that could prompt users with experiences from the past, by the dissection of non-functional requirements, is assumed to be very useful as a support tool by decision-making during the early design stage. Experts assume that this could significantly improve the process of client / designer interaction, and previous to a tender. A pre-condition, that is from great importance for using information by the design of such automated system, is that it should be able to use input information of verified and validated projects from the past. Having such information could determine a certain range in which design decisions are most likely to be configured in, given the results from the past. Section 5.1 treated the evolutionary prototyping process that is initiated for the development of such a prototypical system in which the following operational qualities (system requirements) are accommodated:

- 1) The fundamental operational functions such as Create, Read, Update and Delete (CRUD), need to be accommodated in the system;
- 2) The system needs to be capable to run automated lexical analysis to dissect the linguistic chunks of text, in which product requirements are listed, within a client's brief;
- 3) The system needs to be able to store enriched words, as obtained from the dissected text, by means of state of the art knowledge to formulate the possible meaning(s) of the word within a sentence;
- 4) The systems needs to be able to allocate the enriched words in relation to the subsystems by means of a standard system distributions, such as the NL/SfB;
- 5) The system needs to capture and distinguish the translation of the definitions of obtained words in relation to other acting disciplines, assuming that these definitions can vary;
- 6) The system needs to capture the final specification of the word(s) to formulate a product specification that satisfies the initial requirement.

This functionality is concluded to be from great importance by the development of a more advanced system in which the operational functionality is accommodated.

Main research question: "How can the translation process of non-functional requirements be structured and automated to formulate product specifications in the design process?"

The review of literature has treated both the current curve of knowledge and recent practices in regards to the design process, Systems Engineering, Knowledge Management and Natural Language Constraints in the AEC-industry. The in-house practices revealed, by means of a relatively small sample size, the current practices in regards to the translation of the variety of requirements. The outcome of the observations as merged from both the literature review and the in-house practices contributed to the answer to the main research question.

Requirements manifest their self in different structures and properties. The difference among the types of requirements implicate the level of difficulty for the translation of requirements into product specifications. Science has brought a variety of client requirement specification methodologies as a function of a certain design process. These methodologies and techniques are often adopted from other domains such as Computer Science, Aerospace- Mechanical- Electrical engineering. The tradition of Requirement Management and Engineering contributed to the methodical way in which requirements can be structured before translating them into product specifications. There can be stated that there is a wide range of possible options that treat the translation of client specific requirements as a function of a certain design process. These methodologies and techniques often dictate formal notations of requirements which is crucial. These formal notations employ the formal language in which requirements are treated and communicated within a certain domain which is very crucial given the fact that it can declare the level of ambiguity between all interpreters within a certain process. This is of great importance for the translation of non-functional requirements which are often not quantifiable and therefore more qualitative from nature.

The AEC-industry is adopting these methods and techniques as mentioned. The AEC-domain has also a strong tradition in how client specific requirements can be treated as a function of a design process. Literature provides a variety of methods and techniques in how client specific requirements can be treated within the AEC-industry. However, there are still no formal languages in which the requirements are written and specified. This makes it very hard for interpreters to harmonize the perception of natural language before converting this in raw engineering data. This process is known to be error prone. In this research there has been found that the requirements obtained from clients need to be filtered on what each of the words, combinations of words or combinations of sentences within a chunk of text implies in terms of engineering. This is very hard assuming that there is no conformity between actors within the AEC-domain on what specific concepts should imply.

In this research, there has been chosen to filter words from a sentence by means of tokenization. These tokens basically imply the data structure that a certain word contains. The tokenization of words is used to develop a fundamental formal notation that could be used for processing this Domain Specific Language for computers as an experiment. These tokens are structured by means of a word, definition, class, and a specification. The words can be defined using natural languages such as Dutch, English or Chinese and so on. The definition of words are structured and defined in terms of domain language and shall be expressed by legal, geometrical, structural, building physical, material technical, financial, and aesthetical definition(s). The allocation of these specific definitions of words is related by means of a formal system classification technique, the NL-SfB, that allocates the relation between word, definition, classification, and specification on a system level. This implies that the NL-SfB is also relating the product specifications on a system level. These product specifications are the results that reflect on raw engineering data that implies specific constraints. However, there must be mentioned that it can be a very laborious task to convert qualitative information into quantitative information. Section 7.2 within this research revealed how this could be achieved for a single case. Here, the word 'warm' has been demonstrated to be

converted by means of data analysis in a quantitative RGB color range. Achieving a product specification from a fuzzy text was the initial objective of this research which has been proven by this experiment.

Along this research, there has been found that there are a variety of techniques that can accommodate the introduced methodology within a piece of software. In the development process of this research initiative, the technique that has been used to extract words from a certain sentence written in natural language are hash tables. Without any further due to the characteristics of this technique, there must be mentioned that several ways to Rome could have been chosen. Hash tables have been chosen due to the sake of brevity for evolutionary prototyping and its close practical relation to text mining. Natural Language Processing, and derivatives of this technique, would have been state of the art, but requires a very intense development process. The current technique used proved that automation can be introduced for the translation of non-functional requirements into product specifications. The effectiveness and efficiency of both introduced methods and techniques will be treated within the upcoming chapter on Recommendations.

This page is intentionally left blank

9 Recommendations

This chapter introduces the recommendations that are resulting from this research initiative. The section is divided into a set of recommendations for implementation, and recommendations for future research and development.

9.1 Recommendations for implementation

The first recommendation for the AEC-industry in relation to this research initiative and objective is to start introducing formal languages in which client specific requirements are required to be specified. For example, the Dutch construction industry is currently not using an agreed formal notation that dictates the composition of a client specific requirement. Clients are known to be responsible for descriptive clearance of their demand. However, interpreters might miss or even over do on crucial information. This contributes to the errors that can occur by the interpretation of a requirement by a random interpreter. This process leaves room for additional assumption that one may have for the interpretation of sentences, words, and combinations of both. The variety of these interpretations may lead to misinterpretation of the essence which can lead to ambiguity during business processes. The fact is that these processes occur at the front end of the whole system design process, and that revisions due to the meaning of a requirement might flow in catastrophic unintended events that are hard to revise or even irrevocable. These problems are known to reflect in significant additional costs and delays.

The second recommendation can be given in relation to the first recommendation. If requirements are structured by means of a formal language, then interpretations directly relate to only this notation. This does not imply that ambiguity is not occurring within formal languages, but that the factors that contribute to misinterpretations and ambiguity can be heavily reduced. It is very important that both client and designer understand the design challenges in a harmonized sense given the problem that could occur from this process if not executed correctly. Both clients and designers should ideally use the same formal language in which concepts are expressed by means of natural language prior the verification and validation of requirements.

The third recommendation can be described by the simple fact that client and designer interaction needs to be captured as a function of time by means of formal notations. This could contribute in an efficient and effective way in which information and data can be processed as knowledge. Then, this knowledge can be assumed to be valid if verification and validation has been proven. If knowledge from the past is relying on valid sources, then this can be used to serve mankind. This implies that this knowledge can be captured for use in certain databases which can be consulted for specific queries in future business scenarios. This principle could also contribute to the development of a certain open source database in which client requirements are stored by means of formal notations. It would be very effective and efficient if such open source database would be accessible by both clients and designers since this might reduce the iterative nature of design processes within the AEC-domain. This might reduce delays within the design process given the agreement in thoughts that both parties can have within the early design stages. This is especially

advantageous in competitive environments during the early design stages for tenders where few or little information is available to designers and engineers.

Furthermore, if open source databases are not achievable, then companies are recommended to start developing and maintaining such principles in an internal database since this can provide them managerial edges along the early design process. Capturing knowledge in relation to requirement translation procedures from the past can provide great insight for predictions of system configuration and behavior for future projects. One can state that companies within the AEC-industry are not that familiar with Building Information Management and Information Technology. Then, outsourcing this type of demands or initiating startups as a company holding in regards to this objective can be an obvious options. Design Management is known to be of great importance giving the rise in complexity in demands within the AEC business.

9.2 Recommendations for future research & development

This research initiative towards the translation of ambiguous client requirements into product specifications has showed process improvements and the application of automation by translation of client specific requirements into product specifications. The following summation expresses this research's main findings;

- Improvement of the iterative nature within early design processes;
- Allocation and dissection of the translation processes as a function of the design and business process within the AEC-domain;
- Current threshold and practice in verification, validation, and electronic requirement management;
- Preconditions for automation of the translation of client specific requirements into product specifications;
- Techniques to translate qualitative information into quantitative information obtained from non-functional requirements;
- Framework to filter and structure information and data from client specific requirements into product specifications;
- Implementation of a structured format that could contribute to the structure of future databases in which knowledge due to client specific requirements can be captured;
- Prototype for automated translation of client specific requirements into product specifications;

The preconditions for a system that automates the requirement translation into product specifications has been found. The AEC-industry is known to have executed several trials and demonstrations in which attempts have been initiated to automate this procedure. There has been tried to interpret client specific requirements by means of semi-automated techniques in order to enrich this information in terms of Systems Engineering information and data to support the SE process. There has been found that these attempts were unsuccessful given the following reasons:

- Specific knowledge on lexical analysis, requirement ontologies, standards and semantics are missing within the AEC-industry;
- Manufacturers that take design and design management for their account due to contractual obligations are used to build after design, rather than design and build which results in a lack in functional design competences;
- There is no international or national standard in which concepts are captured, there is no formal language in which requirements can be defined and interpreted by both human and computers;
- Information and data due to client specific requirements and specifications are not delivered, nor captured by means of standard structures.

This research has found that requirement translations into product specifications could be automated if the following input can be provided to feed computer based systems:

- Validated interpretations of requirement with clients and users as obtained from previous projects;
- Availability of a set of dissected requirements, as programmed in previous projects, that are represented in a measurable state;
- Allocation of requirements and objects, as done in previous projects;
- Availability of the total amount of information in regards to the verification procedure, as executed within previous projects.

The eventual way of designing a more advanced and intelligent automated translation system is found along this research initiative. This, especially with the findings during the evolutionary prototyping process. These fundamental system requirements for such advanced systems are summarized as follows:

- Fundamental operational functions such as Create, Read, Update and Delete (CRUD), need to be accommodated within such systems;
- Such systems need to be capable to run automated lexical analysis to dissect the linguistic chunks of text obtained from a client's brief;
- Such systems need to be able to automatically store enriched words by means of a formal notation, as obtained from the dissected text, by means of standardized concepts to formulate the possible meaning(s) of the word within a sentence;
- Such systems need to be able to automatically equate and allocate the enriched words in relation to the subsystems by means of a standard system distributions;
- Such systems need to automatically capture and distinguish the translation of the definitions of the obtained words in relation to other acting disciplines, assuming that these definitions can vary;
- Such systems need to automatically capture the final specification of the word(s) to formulate a product specification that satisfies the initial requirement;
- Such systems need to run on databases that are filled with valid and state of the art knowledge obtained from a variety of projects as delivered in the past.

This page is intentionally left blank

10 References

- 15288, I. (2015). *ISO/IEC/IEEE 15288:2015 Systems and software engineering - system life cycle processes*.
- Abeljaber, R. I. (2017, June 1). *Knowledge management of corporate intranets*. Opgehaald van <http://web.mit.edu/ecom/www/Project98/G4/>
- Alavi, M. &. (2001). Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS quarterly*, 25, 107-136.
- Alharbi, M., & Emmitt, S. (2015). Transferring architectural management into practice: A taxonomy framework. *Frontiers of Architectural research*.
- Argote, L. &. (2000). Knowledge Transfer: A Basis for Competitive Advantage in Firms. *Organizational Behavior and Human Decision Processes*, 82, 150-169.
- BAMinfra. (2008, May 13). *SE-wijzer: Handleiding Systems Engineering*. Opgehaald van http://www.leidraadse.nl/assets/files/images/BN/bestanden/BAM_SE-wijzer.pdf
- Banko, M. &. (2001). Scaling to very very large corpora for natural language disambiguation. *ACL '01 Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*.
- Barney, B. (2017, June 29). *Introduction to Parallel Computing*. Opgehaald van from: https://computing.llnl.gov/tutorials/parallel_comp/
- Belblidia, S. &. (2003). Implicit handling of geometric relations in an existing modeler. *CAADRIA 2003 Conference*, (pp. 613-622). Bangkok, Thailand.
- Bernheim, B., & Whinston, M. (1998). *Incomplete contracts and strategic ambiguity*. 902-932.
- Bilal, M. O. (2016). Big Data in the construction industry: A review of present status, opportunities, and future trends. *Advanced Engineering Informatics*, 30 (2016), 500-521.
- BIMForum. (2017, June 5). *Level of specification*. Opgehaald van www.bimforum.org/lod
- BNA. (2017, July 09). *NL/Sfb-Tabellen inclusief gereviseerde Elementenmethode '91*. Opgehaald van http://www.stabu.org/wp-content/uploads/2015/07/NL-SfB_BNA_Boek_2005-10-90-807626-3-6.pdf
- BNA, N. &. (2017, 05 05). *Standaard bestekbeschrijving DNR-STB 2009*. Opgehaald van http://www.bna.nl/fileadmin/user_upload/Helpdesk/bureauezaken/standaardtaakbeschrijving_2009-def.pdf
- Brown, P., Cocke, J., Pietra, S., Pietra, V., Jelinek, F., Lafferty, J., & Mercer, R. &. (1990). A statistical approach to machine translation. *Computational Linguistics* 16, 79-85.
- Brunton, J., Hellards, R., & Boobyer, H. (1964). *Management Applied to Architectural Practice*. George Godwin.
- BuildingSmart. (2011). *National institute of Building Sciences*. BuildingSMART alliance.
- BuildingSmart. (2013). *NEN-ISO 16739:2013*.
- Campbell, D. (2017, August 8). *Modeling Rules, Architecture week*. Opgehaald van http://www.architectureweek.com/2006/1011/tools_1-1.html

- CB-NL. (2017, May 07). *Concepten Bibliotheek Nederland*. Opgehaald van CB-NL: <http://public.cbnl.org/over-cb-nl>
- Chao-Duivis, M., Koning, M., & Ubink, A. (2013). *A practical guide to Dutch building contracts (3rd edition)*. The Hague: IBR.
- Chen, L., & Luo, H. (2013). A BIM-based construction quality management model and its applications. *Automation in construction* 46, 64-73.
- Chen, M. M. (2014). Big data: A survey. . *Mobile Networks and Applications*, 19, 171-209.
- College, U. d. (2001). *Systems Engineering fundamentals*. MIT.
- Crow. (2005). *Model Basis Overeenkomst, UAV-GC 2005*. Ede: Crow.
- Dave, B. &. (2009). Collaborative knowledge management - A construction case study. *Automation in construction*, 18, 894-902.
- Davis, M. (1985). *Computability and Unsolvability*. Dover Publications.
- Deshpande, A. A. (2014). A framework for a BIM-based knowledge management system. *Procedia Engineering*, 85, 113-122.
- Dimyadi, J., & Amor, R. (2013). Automated Building code Compliance Checking - Where is it at? *Proceedings of CIB WBC 2013*, 172 - 185.
- Dittrich, J. &.-R. (2012). Efficient big data processing in Hadoop MapReduce. . *Proceedings of the VLDB Endowment*, 5.
- Dohmen, M. (1998). *Constraint-Based Feature Validation, PhD thesis*. Delft: Delft University of Technology.
- Donath, D. &. (2007). Constraint-Based Design in Participatory Housing Planning. *eCAADe 2007 Conference, Frankfurt am Main, Germany 'Predicting the Future',*, (pp. 687-694).
- Eadie, R., Browne, M., Odeyinka, H., Mckeown, C., & McNiff, S. (2013). Automation in construction implementation throughout the UK construction project lifecycle: An analysis. *Automation in Construction* 36, 145-151.
- Eastman, C. M. (2011). *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors*. John Wiley & Sons. .
- Eggink, D., & Gross, M. &. (2001). Smart Objects: Constraints and Behaviors in a 3D Design Environment. *eCAADe 2001 Conference, Helsinki, Finland 'Architectural Information Management*, (pp. 460-465).
- El. Reifi, M. H. (2013). Perceptions of Lean Design Management. *Architectural Engineering and Design Management* 9. 195-208.
- Elsberg, & Daniel. (1961). Risk, ambiguity and the savage axioms. *Quarterly journal of economics*, 643-669.
- Engineering, W. L. (2007, June 1). *Leidraad voor Systems Engineering binnen de GWW-sector*. Opgehaald van <http://www.leidraadse.nl/downloads>
- Environment, M. o. (2005, May 21). *Handreiking Functioneel Specificeren*. Opgehaald van http://www.coinsweb.nl/downloads/Handreiking_functioneel_specificeren.pdf
- Fayyad, U. P.-S. (1996). From data mining to knowledge discovery in databases. *AI magazine*, 17.

- Frappaolo, C. &. (1997). Knowledge management: From terra incognita to terra firma. *The knowledge management yearbook 1999-2000*, 381-388.
- Gehry, F. (2017, 02 18). *Frank Gehry Teaches Design & Architecture | Official Trailer*. Opgehaald van [www.youtube.com: https://www.youtube.com/watch?v=Az-m56vUjgw](https://www.youtube.com/watch?v=Az-m56vUjgw)
- Glinz, M., & Wieringa, R. (2007). Guest editors' introduction: Stakeholders in requirements engineering. *IEEE Software* 24, 18-20.
- Grant, S., Kline, J., & Quiggin, J. (2009). *A matter of interpretaion: bargaining over ambiguous contracts*. Bond University.
- Halman, J., & Voordijk, J. &. (2008). Modular Approaches in Dutch House Building: An Exploratory Survey. *Housing Studies* 23, 781-799.
- Halpern, J. Y., & Kets, W. (2015). Ambiguous language and commong priors. *Games and Economic behaviour*, 171-180.
- Harris, P. W. (2017, December 12). *HERACLITUS The Complete Philosophical Fragments*. Opgehaald van Community Middelbury: <https://community.middlebury.edu/~harris/Philosophy/Heraclitus.html>
- Hull, E., Jackson, K., & Dick, J. (2006). Requirements engineering. *Requirements Engineering* 13.
- IBM. (2016, October 18). *The Four V's of Big Data*. Opgehaald van <http://www.ibmbigdatahub.com/infographic/four-vs-big-data>
- INCOSE. (2007). *Systems engineering handbook: A guide for systems life cycle processes and activities*. C. Haskins, Ed.
- INCOSE. (2015). *Systems Engineering handbook: A guide for system life cycle processes and activities*. Hoboken, NY: John Wiley and sons.
- Jahan, A., & Edwards, K. L. (2013). *Multi-criteria decision analysis for supporting the selection of engineering materials in product design*. Amsterdam: Elsevier.
- Jallow, A., Demian, P., Anumba, C., & Baldwin, A. N. (2017). An enterprise architecture framework for electronic requirements information management . *International Journal of Information Management*, 455-472.
- John, G. H. (1997). *Enhancements to the data mining process, PhD thesis*. Stanford, USA: Stanford university.
- Junnarkar, B., & Brown, C. (1997). Knowledge Management: An emerging discipline with a long history. *Journal of Knowledge Manamgent* , 142-148.
- Kelleners, R. (1999). *Constraints in object-oriented graphics, PhD thesis*. Eindhoven: Eindhoven University of Technology.
- Kim, T. W., Kim, Y., Cha, S. H., & Fisher, M. (2015). Automated updating of space design requirements connecting user activities and space types. *Automation in construction*, 102-110.
- King, M. (1983). *Parsing Natural Language*. London: Academic Press Inc. Ltd.
- Kiviniemi, A. (2005). Requirements management interface to building product models. *VTT Publications*.
- Knotten, V., Svaluestuen, F., Hansen, G., & Laedre, O. (2015). Design management in the building process - A review of current literature. *Procedia Economics and Finance* 21, 120 – 127 .

- Knotten, V., Svaluestuen, F., Hansen, G., & Laedre, O. (2017). Building design management – key success factors. *Architectural Engineering and Design Management*, 479-493.
- Knuth, D. (1964). *Backus normal form vs. Backus Naur form, Letter to the editor*.
- Kocher, M., Lahno, A., & Trautmann, S. (2017). Ambiguity aversion is not universal. *European Economic Review*.
- Kock, N. F. (1997). The nature of data, information and knowledge exchanges in business processes: implications for process improvement and organizational learning. *The Learning Organization*, 4, 70-80.
- Laney, D. (2001). *3D data management: Controlling data volume, velocity and variety*. Stamford: META Group Inc. .
- Leeuwen, J., Jessurun, A., & de Wit, E. (2004). "The Digital Dormer - Applying for Building Permits Online". (pp. 355-361). Rotterdam: Balkema.
- Leijen, D. &. (2001). *Parsec: Direct Style Monadic Parser Combinators For The Real World*.
- Liebich, T., Adachim, Y., Forester, J., Hyvarinen, J., Richter, S., & Chipman, T. (2013). Industry Foundation Classes release 4 (IFC4) documentation.
- Lu, W., Fung, A., Liang, C., & Rowlinson, S. (2017, June 1). *Demystifying construction project time-effort distribution curves: a BIM and non-BIM comparison* Weisheng. Opgehaald van <http://ascelibrary.org/doi/abs/10.1061/9780784413517.034>
- Lu, W., Fung, A., Peng, Y., & Liang, C. R. (2014). Cost-benefit analysis of BIM implementation in building projects through demystification of time-effort distribution curves. *Building and Environment* 82, 317-327.
- Malsane, S., Matthews, J., & Lockley, S. L. (2015). Development of an object model for automated compliance checking. *Automation in construction* 49, 51-58.
- Manning, C. &. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge: The MIT Press.
- Marchant, A. (2010). Obstacles to the flow of requirements verification. *Systems Engineering* 13.
- Moonen, L. (2016). *Improving the design process: The implications of automated verification of client specific requirements using semantic web standards and rule checking techniques*. Eindhoven: Eindhoven University of Technology.
- Nederland, B. (2014, May 13). *Praktische Leidraad voor geïntegreerd samenwerken met de UAVgc in de woning- en utiliteitsbouw*. Opgehaald van <http://www.bouwendnederland.nl>
- Ng, A. (2017, June 23). Opgehaald van <http://www.andrewng.org/publications/>
- Niemeijer, R. A. (2011). *Constraint specification in architecture: A user-oriented approach for mass customization*, PhD thesis. Eindhoven: Eindhoven University of Technology.
- Normalisatie-instituut, N. (1993). *NEN 2574 - Construction drawings. Arrangement of data on building drawings*.
- Parr, T. &. (1995). ANTLR: A predicated-LL(k) parser generator. *Software: Practice and Experience* 25, 789-810.
- Pels, e. a. (2013). Systems Engineering as a first step to effective use of BIM. *Product lifecycle management for society*, 651-662.
- Pozzali, A. &. (2015). Cognition, Types of "Tacit Knowledge" and Technology Transfer. In *Cognitive Economics: New Trends*. , 205-224.

- Prorail. (2015, May 15). *Handboek Systems Engineering (SE) - Overzicht in processen, informatie en technieken*. Opgehaald van <http://www.leidraadse.nl/downloads>
- Rekvelde, Y. (2016). *BIM Based Cost Estimation Knowledge Management*. Eindhoven: Eindhoven University of Technology.
- Rijkswaterstaat. (2017, June 1). *Procesbeschrijving systems engineering voor RWS projecten*. Opgehaald van <http://www.leidraadse.nl/downloads>
- Samset. (2008). *Prosjekt i tidligfasen: valg av konsept*. Trondheim.
- Schaap, H., Bouwman, J., & Willems, P. (2017, June 3). *COINS-referentiekader voor functioneel specificeren*. Opgehaald van www.coinsweb.nl
- Scheithauer, D., Esep, I., & Forsber, K. (2013). *V-Model Views*.
- Schneider, F., & Berenbach, B. (2013). A literature survey on international standards for system requirements engineering. *Procedia computer science*, 16, 796-805.
- Scott, R., & Trantis, G. (2006). Anticipating litigation in contract design. *Yale Law Journal*, 814-879.
- Shishko, R., & Aster, R. (2007). *NASA systems engineering handbook*. Harvard.
- Short. (2011). *The law Library and Contract book*. Opgehaald van http://inform7.com/learn/eg/bronze/source_12.html
- Siddharth, L., & Sarkar, P. (2017). A Methodology for Predicting the Effect of Engineering Design Changes. *Procedia CIRP* 60, 452-457.
- Sipser, M. (1996). *Introduction to the Theory of Computation*. PWS Pub. Co.
- Solihin, W., & Eastman, C. (2015). A knowledge representation approach to capturing BIM based rule checking requirements using conceptual graph. *Proc. of the 32nd CIB W78 Conference*. Eindhoven.
- Solihin, W., & Eastman, C. (2015). Classification of rules for automated BIM rule checking development. *Automation in construction* 53, 69-82.
- Sparrius, A. (2014). The life cycle of a requirement. *INCOSE international symposium* 42, 417-436.
- Spinellis, D. (1999). Reliable software implementation using domain-specific languages. *ESREL, 10th european software conference on safety and reliability 'ESREL, 10th european software conference on safety and reliability'*.
- Thompson, C., & Califf, M. &. (1999). Active learning for natural language parsing and information extraction. *In Proc. 16th International Conf. on Machine Learning (1999)*, (pp. 406-414).
- Tilley, P. A. (2005). Lean Design Management- A New Paradigm for Managing the Design and Documentation Process to Improve Quality. *Proceedings of the IGLC-13*.
- Visser, A. (2011). *Handboek specificeren*. Ede: CROW.
- Walker, D., Davis, P., & Stevenson, A. (2017). Coping with uncertainty and ambiguity through team collaboration. *International Journal of Project Management* 35, 180-190.

- Walraven, A., & de Vries, B. (2009). From demand driven contractor selection towards value driven contractor selection. *Construction management and Economics* 27, 597-604.
- Wang, W. (2011). Ambiguity in language. *Korea Journal of Chinese Language and Literature* 1, 5-17.
- Ward, J. S. (sd). *Undefined by data: a survey of big data definitions*. Fife, Scotland: University of St. Andrews .
- Witten, I. (2005). *Text mining, Practical handbook of internet computing*. Boca Raton, Florida.: Chapman & Hall/CRC Press.
- Wix, J., Nistbet, N., & Liebtich, T. (2008). Using Constraints to Validate and Check Building Information Models. (pp. 467-476). France: CRC Press.
- Zhang, C., Beetz, J., & Weise, M. (2015). Interoperable validation for IFC building models using open standards. *Journal of Information Technology in Construction*, (pp. 24-39). Eindhoven.
- Zhang, Y., Xiaofang, L., Zhao, Y., & Hong-chao, Z. (2015). An ontology-based knowledge framework for engineering material selection. *Advanced Engineering Informatics*, 985-1000.

11 Appendices

11.1 Appendix A: Program design

Within this chapter, a detailed description of the system design will be given to clarify reasoning and decision making for software development. The actual codes of the program are attached as Appendices of this report, which might be very useful by reading this chapter. The program is written by means of Java, which is a formal programming language. Java is known by the software industry to be a high-level, object oriented programming language that uses a compiler. It is mostly recognize for its, sometimes overdone, user-friendly environment which often causes major delays. However, these delays have significantly decreased since Java 7 and 8 and is nowadays known to be equal in speed compared to other object oriented languages such as C++. The average runtime of the Java compiler is way faster in comparison to other programming languages. Still, the decision on the implementation of this programming language was mostly related to the ease by the creation of a GUI given the existing Java libraries (especially the javax.swing and java.awt libraries) and personal knowledge and preferences. The developed program consists of ten classes, containing the following classes, including the package name: building.BufferedReaderPlus, building.Word.Def, building.Generate.Design, and building.Advisor, building.log.Log, building.editor.Def, building.editor.DefFileEditor, building.editor.Sort, building.editor.WordCell, building.editor.WordCellGroup.

How these classes are designed, structured, and related as a function of the initial system requirements is described in a practical way within the upcoming paragraphs.

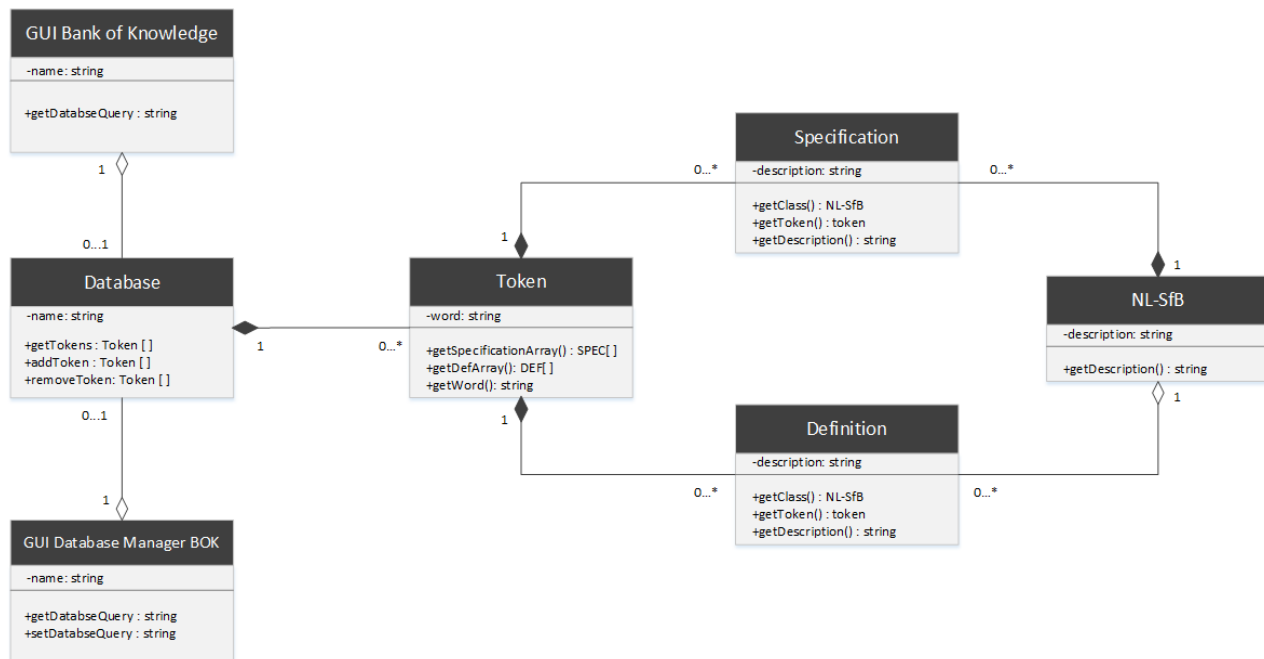


Figure 57: UML class diagram of the system

11.1.1building.BufferedReaderPlus

The aim of this class is to provide an easier environment for handling the IO-operations of parsing a file. Please note that this is a very general class written for multiple cases, so many functions won't be used in this case. All function keeps track of the line counter of the file for debugging purposes. The following gives a short description of each function within the program its code:

1.0 Constructor

The constructor is public and can accept four variables:

1. **java.io.Reader reader**
To use this class as a wrapper class for inputstreams.
2. **int commentType**
To determine the comment type that is used for parsing the file (see 1.2 *readProccessedLine*) for more info about this).
3. **String singleCommentType**
This determines the custom single line comment (see 1.2 *readProccessedLine* for more info about this).
4. **boolean isCsv**
To determine whether the file should be read as a CSV file. Alters the functionality of the following function: 1.2 *readProccessedLine*. This allows the use of the following functions: 1.4 *readCSVCell*, 1.5 *readCSVLine*.

1.1 public String readLine()

Reads a line from a file without enchantments. Uses the *readLine* method from its parent (java.io.BufferedReader). If you only want to use this function from this class, then it's better to use "java.io.BufferedReader" instead.

1.2 public String readProccessedLine()

This function reads a line and ignores empty lines and comments. There are five comment modes possible:

1. No comment type (only ignoring empty lines);
2. The character "#" denote the beginning of a single line comments. Therefore, everything after "#" is ignored
3. The characters "/" denote the beginning of a single line comments, the characters "/*" denote the beginning of a multiple line comments and the characters "*/" denote the end of a multiple line comments. Therefore, everything after "/" and between "/*" and "*/" is ignored;
4. The characters ":\\" denote the beginning of a single line comments;
5. A custom String Object denotes the beginning of a single line comments;

The type of comments can only be set once on creation of the object (see 1.0 *Constructor* for more info). Also, if and only if the source file type is set to CSV, the trailing semi-colons are ignored. This function uses the function 1.1 *readLine* to read the data from the file.

1.3 public static String removeCSVTrailings(String line)

This function removes all trailing semi-colons from the input line and outputs that line. Because CSV files end with multiple semi-colons at the end of each line and it is annoying to read empty cells several times just before you reach an EOL.

1.4 public String readCSVCell(boolean processed, {boolean lineBlock})

This function can only be used if the source file type is CSV. It is assumed that semi-colons are used as separators (most used separator for CSV files). The boolean “processed” determines whether a processed line or a not processed line should be read (see *1.1 readLine* and *1.2 readProcessedLine*). The boolean “lineBlock” determines whether the function should check for a new line if the end of the line was reached. The problem that arises concerning the line counter is solved by temporarily storing the remaining line in a global variable. This function will also work without passing the variable “lineBlock” (achieved through overloading functions). Default is false. This function is recommended for single-cell data processing.

1.5 public ArrayList<String> readCSVLine (boolean processed, {boolean lineBlock})

This function does essentially the same as 1.4, but instead of reading one cell, it reads all cells on a line and puts them in an ArrayList. This function is recommended for single-cell line data processing.

1.6 public void mark(int readAheadLimit)

This function makes the present location in the stream. After invoking “reset()” you jump back to this location. Jumping back to this location cannot be guaranteed after “readAheadLimit” characters were read from the stream. Uses the parent function *mark(int readAheadLimit)* to handle the IO-operations concerning the file.

1.7 public void reset()

This functions resets to the last marked point in the stream. Uses the parent function *reset(int readAheadLimit)* to handle the IO-operations concerning the file.

1.8 public int getLineCounter()

This function returns the line counter of the file.

1.9 public int read()

Deprecated function

Unsupported operation. This function Throws an UnsupportedOperationException when called. This function is used to prevent unnoticed reading from the file via the parent class. This should not be used.

1.10 public static <A, B extends A> A[] listToArray(List list, Class classValue)

A generic function that converts the given List object “list” to an array of the same type or a child of that type.

1.11 public static <A, B extends A> ArrayList<A> arrayToArrayList(B[] array, Class classValue)

A generic function that converts the given array “array” to an ArrayList object of the same type or a child of that type.

11.1.2 building.log.Log

The Log class consists only of static methods which are used to easily create a log file including timestamps, nice alignments and can be used for the most common classes. It has only two functions:

2.1 public static void write(<?> text, {boolean full}, {boolean showDate})

This function writes a line in the log file. The variable “text” can be one of the following primitive data types/classes:

- boolean
- characters
- int
- double
- Exception
- String

The boolean variable “showDate” determines whether a time-stamp or whitespaces must be shown at the beginning of the line. This variable is optional for all of the above classes. The boolean variable “full” can only be used for exceptions. It determines whether the full exception or the shorter version should be used.

2.2 public static void clear()

Erases all data from the file. Then prints the current date and time on the first line.

11.1.3 building.GenerateDesign

This class has only one function and is used as a wrapper code for a function in “building.Advisor” to create a better overview of the code.

3.1 protected static void generateDesing(HashTable<String, WordDef> definitions, String text, int mostWords)

In this function is the input text “text” processed into three outputs using the library “definitions”. Each output is written to a file. This is done to prevent an overflow in the output data (you need more than $2^{31} - 1 - 1 = 2.147.483.646$ characters for that). It’s used to split the input into words, determine the type of each word (e.g. noun, article, etc.), connect the words in the sentence (e.g. an adjective tells something about a noun), determines meaning of that sentence and finally converts that to plain definitions. Given the initial system requirements it is assumed to be favourable, both technically and in usability, to attach more definitions (knowledge, information) to a word over listing the same word with different definitions multiple times, so every definition of a word is processed only once.

Pseudocode:

```
01 Create output writers "out1", "out2" and "out3"
02 For every type "type":
03 | Create new dynamic array "wordSeen"
04 | For every line "line" of the input:
05 | | For every word "word" in "line":
06 | | | Search "word" in the definitions and put them in array "defs"
07 | | | For every definition "def" in "defs":
08 | | | | If "wordSeen" contains "def":
09 | | | | | discard "def" from "defs"
10 | | | | Else:
11 | | | | | add "def" to "wordSeen"
12 | | | | +
13 | | | +
14 | | | If first iteration of type "type":
15 | | | | Print the String representation of "type" to "out2" and "out3"
16 | | | +
17 | | | If "type" == OTHER:
18 | | | | Print "word" and all definition of word of type "type" to writer
"out1"
19 | | | Else:
20 | | | | Print "word" and all definition of word of type "type" to writer
"out2"
21 | | | +
22 | | | Print all values of word "word" of type "type" to writer "out3"
23 | | +
24 | +
25 +
26
27 Log errors and warnings
28 Return
```

11.1.4 building.WordDef

This class is used as an easy way of storing and retrieving word definitions. Note that the actual word is not stored in this class. This class also contains data fields for easy access and iteration of the definitions.

4.0 Constructor

The constructor is public. This function creates a new definition of a word using word definitions, NL-SfB class definitions (both named "def" in the code) and NL-SfB class specifications (named "value" in the code). Each of them must be an array such that each definition type is defined. Each element of the array contains an *ArrayList* containing all sub-definitions of a specific type.

1. **ArrayList<String>[] newDefs**

This definition stores the word definitions and NL-SfB class definitions of a word.

2. **ArrayList<String>[] newValues**

This definition stores the NL-SfB class specifications of a word.

4.1 private void setDef(ArrayList<String> newDefs, int I)

Deprecated function

Replaces the word definitions or a NL-SfB class definition of a word with the given definitions. This function should not be used, or with extreme care. Might cause synchronization issues. “i” must be one of the definition types.

4.2 public ArrayList<String> getDef(int i)

Returns the word definition, or a definition of the ith NL-SfB class.

4.3 public ArrayList<String>[] getAllDef()

Returns all word definitions and NL-SfB class definitions.

4.4 public ArrayList<String> getValues(int i)

Returns specification of the ith NL-SfB class type. “i” must be one of the predefined values of this class.

4.5 public static String getTypeString(int num)

Returns the String representation of the word definition, or the numth NL-SfB class. Returns “ - [TYPE NOT DEFINED] - ” if there was no match.

4.6 public static int getTypeFromString(String text)

Returns the integer representation of the word definition or NL-SfB class denoted by the input text. Returns the definition “NONE” if there was no match.

4.7 public static String getDevValueString(int num)

Returns the String representation of num, where num denotes either the field “DEF” or the field “VALUE”. Returns “ - [TYPE NOT DEFINED] - ” if there was no match.

4.8 public String toString()

Overrides the *toString()* method from the Object class for debugging purposes. Now, when a String representation of this class is made, all definitions and specifications are printed correctly.

11.1.5 building.Advisor

This class creates the GUI for the application and should be created when the user wants to run the program. A hash table is used for storing the definitions which will be used for translating the input (see 3). This choice is made because each key (the word to search for) must have at least one value (the definition). Also when we take a look at the running times for adding elements (add key and value) and searching (given: key, search: value), then this is simply the best option since it does both actions in constant time per element (using the simple uniform hashing assumption). Thus for adding it depends only linearly on the number of input definitions in the file and for searching it depends only on the number of words of the input text. The GUI of this application consists of multiple parts:

- **Input**

This part consists of the text input block. The user can input the text to convert here.

- **Output**

This part consists of three output boxes. The one below the input shows the word definitions. The one in the middle shows the NL-SfB class definitions. The one on the right shows the NL-SfB class specification. Hyperlinks can be used in the output fields by defining them as html hyperlinks. For this function the *HyperlinkListener* (see 5.18) is used.

- **Buttons**

This part consists of 5 (command) buttons:

- **Lexical analysis button**

Opens a webpage to “demo.ark.cs.cmu.edu/parse”, where the input can be analysed on structure. The data from the input field is automatically filled in and generated on the site. Uses *ActionListener* (see 5.17).

- **Generate design button**

Generates the design by using *ActionListener* (see 5.13).

- **Open output file buttons (1, 2, 3)**

Opens one of the output files by using *ActionListener* (see 5.14).

- **Reload definitions button**

Reloads the definitions by using *ActionListener* (see 5.15).

- **Open definition file button**

Opens the definition file by using *ActionListener* (see 5.16).

This class contains a lot of functions and sub-classes. All of these will now be discussed:

5.0 Constructor

The constructor is public. This function creates the class and starts up the application. It only invokes the functions *createGUI* (see 4.3) and *readDef* (see 4.6).

5.1 private synchronized void generateDesign()

This method is used to invoke the wrapper function *generateDesign* (see 3.1).. First it checks if all definitions were read (since the function *readDef* creates a multi-thread process which might cause some synchronisation issues here, see 4.6). Then invokes the wrapper-function. After the function has terminated normally and under the assumption that the output is written in the predefined output files, reads these output file and copies their contents to the output panels of the application. This method has the “synchronized” keyword. Therefore it is multi-thread safe (only one thread is allowed at the same time).

Pseudocode:

```
01 Remove button listeners
02 Create the hash table "definitions"
03 Create reader "in"
04 Create "defType" = OTHER
05 Create "isDefOrValue" = DEF
06 Create arrays[num of types] of dynamic lists "def" and "value"
```

```

07
08 While EOF not reached:
09 | Read all cells and put them in order in array "line"
10 |
11 | For every element "cell" in "line", start with 2nd element
12 | | If "cell" denotes a def/value change:
13 | | | Set "isDefOrValue" to resp. DEF or VALUE
14 | | | Continue
15 | | +
16 | | If "cell" denotes a def type change:
17 | | | Set "defType" to resp. one of the def types
18 | | | Continue
19 | | +
20 | |
21 | | If "isDefOrValue" == DEF:
22 | | | Add "cell" to "def"["defType"]
23 | | Else:
24 | | | Add "cell" to "value"["defType"]
25 | | +
26 | +
27 | Put value {"def", "value"} with key "line"[0] in hash table
  "definitions"
28 +
29
30 Log errors and warnings
31 Restore button listeners
32 Return

```

5.2 private void createGUI()

Creates the GUI of the application.

5.3 public String generateWebpage()

Generates the webpage URL for the lexical analysis.

5.4 private void addAllButtonListeners()

This method adds the *actionListeners* to the buttons. This method is used as an abbreviation for writing each of these lines separately. It also prevents hard to trace bugs which might occur when an extra *actionListener* is introduced and it should be added on multiple location but one of them is forgotten.

5.5 private void removeAllButtonListeners()

This method removes all *actionListeners* from the buttons. The general idea behind this method is the same as in 4.4.

5.6 public void readDef()

This function acts as a wrapper function. It creates a new thread from where it calls the synchronized function *readDefinitions* (see 5.7). "synchronized" means that only one thread at a

time is allowed to execute the function. Other threads must wait till that one has finished. The idea of using multiple *Threads* is that the application will never noticeably stop functioning, considering that the loading time of the definitions takes some time to complete.

5.7 private synchronized void readDefinitions()

This function reads the definitions from the input file and puts them in the hash table (see intro 4). It reads definitions if and only if (1) they were not yet read before or (2) a request was made to re-read them. At first it removes all *ActionListeners* from the buttons (see 4.5) to ensure that no other actions can be invoked (e.g. a request for re-reading the definitions). Then it reads the definitions from the CSV file using *readCSVLine* (see 1.4) of the class *building.BufferedReaderPlus*, and puts the data in two array of *ArrayList<String>*'s, containing the definitions and the values of that word. This part has been designed such that empty cells are ignored and the total number of cells can differ. Also, for each definition can an arbitrary number definitions and values be assigned by putting a star (*) and the String representation of the definition type in the cell before the definitions. To change from definitions to values of a definition type, put a double star (**) with "value" written after it in the cell before the value definitions of that definition type. Then a new *WordDef* is created using two *ArrayList<String>*'s. This *WordDef* is then stored in a *Hashtable<String, WordDef>*, using the word of which the definitions belong to as key. After putting the definitions in the hash table, this method logs some data about the errors and/or warnings it had during reading of the definitions. Finally it adds the *ActionListeners* back to the buttons (see 5.4) and sets the flag that all definitions are now read to true.

5.8 public static boolean openFile(File file)

Opens the given *File* file with the default application. Logs an error if no application for that file type was registered or the desktop was not supported.

5.9 public static boolean openWebpage(String stringURI)

Overloaded function for the *openWebpage* (see 5.10) function. Converts the string into an URI and passes it on to function 5.10.

5.10 public static boolean openWebpage(URI uri)

Opens a webpage with *URI* uri as destination (the same concept as the search bar of your browser). Returns true if the website could be opened and no Exceptions occurred. Otherwise return false and log the Exception.

5.11 DocumentListener textAreaDocListener

Creates a new *DocumentListener* object which regulates a default text for an *textArea*. Overrides the following functions from the *DocumentListener* class:

5.11.1 changeUpdate(DocumentEvent e)

This method is called when any change occurs. No action is taken.

5.11.2 insertUpdate(DocumentEvent e)

This method is called when one or more characters are inserted. This function checks if this was

the first character entered by the user. If so, remove all default text except for all characters which were inputted by the user. Also colours the text black in that case. Here is assumed that the text is inserted at the beginning of the text area (is ensured by 5.12). This function temporarily removes itself from the caller and is executed on another thread because it manipulates data which will otherwise lead to infinite recursive calls to itself.

5.11.3 removeUpdate(DocumentEvent e)

This function is called when one or more characters are deleted. This function checks if there are still characters left. If none left, put the default text back and colour it grey. This function temporarily removes itself from the caller and is executed on another thread because it manipulates data which will otherwise lead to infinite recursive calls to itself.

5.12 CaretListener textAreaCaretListener

Creates a new *CaretListener* object. Only overrides the *caretUpdate* function from the *CaretListener* class. This function sets the position of the cursor to the beginning of the text area. This function temporarily removes itself from the caller and is executed on another thread because it manipulates data which will otherwise lead to infinite recursive calls to itself.

5.13 ActionListener generateButtonActionListener

Creates a new *ActionListener* object. Only overrides the *actionPerformed* function from the *ActionListener* class. This function is called when a certain action is performed by its caller. In this case, since its only caller is a *JButton* object, it is a button press. This method requests to generate the output by calling the method *generateDesign* (see 5.1).

5.14 ActionListener openOutputFileActionListener

Creates a new *ActionListener* object. Only overrides the *actionPerformed* function from the *ActionListener* class. This function is called when a certain action is performed by its caller. In this case, since its only caller is a *JButton* object, it is a button press. This method opens the output file by calling the method *openFile* (see 5.8). Depending on which button was pressed, opens output file 1, 2 or 3.

5.15 ActionListener editDefActionListener

Creates a new *ActionListener* object. Only overrides the *actionPerformed* function of the *ActionListener* class. This function is called when a certain action is performed by its caller. In this case, since its only caller is a *JButton* object, it is a button press. This method creates a new *DefFileEditor* (see 5.5.6) and saves the definition file to a backup file (maximal 9 different backup files can exist) if no *DefFileEditor* exists. Else set the previous created *DefFileEditor* visible. In any case, add the *windowAdapter closeWindow* (see 5.21) to the mainframe.

5.16 ActionListener openDefFileActionListener

Creates a new *ActionListener* object. Only overrides the *actionPerformed* function of the *ActionListener* class. This function is called when a certain action is performed by its caller. In this case, since its only caller is a *JButton* object, it is a button press. This method opens the definition

file by calling the method *openFile* (see 5.8).

5.17 ActionListener openParserActionListener

Creates a new *ActionListener* object. Only overrides the *actionPerformed* function of the *ActionListener* class. This function is called when a certain action is performed by its caller. In this case, since its only caller is a *JButton* object, it is a button press. This method opens the webpage to the online parser when the parser button is pressed. Also inputs the data from the input field in the input of the online parser. Uses the function *generateWebpage* (see 5.3) to generate the link for the webpage and the function *openWebpage* (see 5.9) to open the webpage.

5.18 HyperlinkListener openHyperlinkListener

Creates a new *HyperlinkListener* object. Only overrides the *hyperlinkUpdate* function from the class *HyperlinkListener*. This function is called when an action is performed on a hyperlink i.e. the hyperlink was clicked. Uses the function *openWebpage* (see 5.9) to open the webpage.

5.19 ComponentListener windowResizeListener

Creates a new *ComponentListener* object. Only overrides the *componentResized* function from the *ComponentListener* class. This function is only called when the size of the main frame is resized. Uses the function *update* (see 5.20) to update the GUI of the main frame.

5.20 public void update()

This function is called when the dimensions of the main frame have changed, or when a GUI update is required. In this case. This method sets all fields according to the new dimensions for a smooth look.

5.21 WindowAdapter closeWindow

Creates a new *WindowAdapter* object. Only overrides the *windowClosing* function from the *WindowListener* class. This function is called when the application is closed. This class is only used when the *DefFileEditor* is visible and asks if the user really wants to close the application since any unsaved changes in the editor will not be saved. Also waits till all writers of the editors have stopped writing.

5.22 public void updateMaxWordLength(int mostWordsNew)

This function is only called by the *DefFileEditor* when a new entry was saved. This method updates the maximal number of words in a definition. Note that there is no function for deleting entries, because it does not affect correctness of the application and is calculated correctly when the program is restarted.

5.23 ActionListener showHideOutput_1AL

Creates a new *ActionListener* object. Only overrides the *actionPerformed* function of the *ActionListener* class. This function is called when a certain action is performed by its caller. In this case, since its only caller is a *JButton* object, it is a button press. In this case, it is the show/hide button for the word enrichment field. This function changes the text of the show/hide button,

adds/removes the word enrichment field and updates the GUI of the application.

5.24 public void notifyClosedDefEditor()

This function is only called by *DefFileEditor* when it closes. This method removes the *windowAdapter closeWindow* (see 5.21) from the main frame, sets the *DefFileEditor* not visible and sets the default close operation of the application to exit on close.

11.1.6 building.editor.DefFileEditor

This class is used to create the GUI and to handle all actions concerning the IO between the advisor, the editor and the user. To prevent multiple not updated versions of library, there is used exactly one instance of the *Hashtable* containing all definitions, but multiple pointers pointing to it.

When the editor is created, a backup file is made to prevent unexpected data loss in case in the most common types of system or application failures, or human errors.

The GUI consists three main panels:

- **Layout bar**

With the buttons on this bar you can add the html code for bold, italic, underline and a weblink for the definitions. See 7.12 for more info.

- **Token configuration**

This consists of three parts:

- **Input word**

The word the token defines is shown/edited here

- **Definition panel**

In this panel appear all definitions of the input word. You can add a definition by pressing the “Add definition” button. Every definition consists of four parts:

- **Definition or NL-SfB class**

Here you can select whether you want to define a definition of the word or a class definition.

- **Type or class definition/specification**

If you selected “definition”, you can choose the type of definition of the word. If you selected a NL-SfB class, you can choose for either a class definition or a class specification.

- **Input field**

The data you want to put in the definition.

- **Checkbox**

To select the definition.

- **Buttons**

There are five buttons:

- **Edit entry**

This button loads the entry selected in the database panel from the (local) library to the definition panel. Note that pressing this button will erase

unsaved data from the previous edited entry.

- **Save entry**
Saves the currently displayed entry to the library. Note that this does NOT save the entry to the definition file, but only in the cache of the application. This means that you can use it for this session only.
- **Save changes**
Saves the current library to the definition file. Note that this does NOT include the unsaved entry currently displayed, so use the save entry before saving the changes.
- **Del selected def(s)**
Deletes all selected definitions. A definition can be selected by pressing the checkbox on the right of that entry.
- **Delete entry**
Deletes the entry selected by the database panel. Note that this does NOT delete the entry currently displayed in this panel.

- **Database panel**

This panel consists of two parts:

- **Search bar**
Here you can enter a search term and press the “Search” button to decrease the amount of entries you have to look at before finding the right one. Note that only the first part of an entry is compared to the search term, so searching for “bc” will not yield the entry “abc”.
- **Database entries**
In this part, the words that are defined in the database are displayed in two columns in alphabetical order (from left to right, top to bottom, other characters first, then numbers, then letters).

This class contains a lot of functions and sub-classes. All of these will now be discussed:

6.0 Constructor

The constructor is public. This function creates the GUI of the editor, sets some initial values for variables and adds listeners to the buttons. It takes the following as input: *Advisor* advisor, *File* defFile, *File* bacFile, *Hashtable<String, WordDef>* defs, *int* locX, *int* locY “advisor” denotes the caller class. This variable is passed to create a smooth communication between the two classes. “deFile” denotes the file which stores the definitions. “bacFile” denotes the file where the backup file should be stored. “*Hashtable<String, WordDef>*” denotes the library, “locX” and “locY” denote the initial x- and y-coordinates of the editor main frame. This constructor uses the *invokeLater* function from *SwingUtilities* because otherwise the generation of the GUI might cause the application to halt for a certain amount of time.

6.1 createBACThread()

Creates a new Thread that makes a copy of the current definition file and writes it to the backup file.

6.2 private void addDef(Def def)

Adds a definition to the definition panel, adds a *CaretListener* to def to keep track of the last location of the cursor and revalidates “def” because the *JComboBox* has some issues with correctly displaying after being added to a visible component (known issue in Java).

6.3 private void removeDef(Def def)

Removes a definition from the definition panel. Also removes the *CaretListener* to keep track of the cursor and if this field was the last one visited of all *Defs*, set that pointer to *null*.

6.4 private void clearDefs()

Deletes all definitions from the definition panel. Recursively calls the function *removeDef* (see 5.3).

6.5 private void addSearchResult(String name)

Adds an entry to the database entry panel, sets the size and location for the new *WordCell* (see 5.5.9) and adds it to the *WordCellGroup* (see 5.5.10).

6.6 private void removeSearchResult(WordCell wc)

Deletes an entry from the database entry panel and from the database. Also removes the *WordCell* from the *WordCellGroup*.

6.7 private void hideSearchResult(WordCell wc)

Removes an entry from the database entry panel and removes the *WordCell* from the *WordCellGroup*.

6.8 ComponentAdapter windowResizeListener

Creates a new *ComponentAdapter* object. Only overrides the *componentResized* function from the *ComponentListener* class. This function is called when the main frame of the editor is resized. Only calls the function *updateGUI* (see 6.9) to update the GUI of the editor.

6.9 public void updateGUI(Boolean full)

Updates the GUI of editor. “full” denotes whether a full update or an intermediate update should be done.

6.10 ActionListener buttonActionListener

Creates a new *ActionListener* object. Only overrides the *actionPerformed* function of the *ActionListener* class. This function is called when a certain action is performed by its caller. In this case, since its only caller is a *JButton* object, it is a button press. This function is executed on another Thread to prevent the program from blocking and reducing the use of the Swing Thread. This listener is added to all buttons of the editor and handles their actions as described in the introduction of 5.5.6.

6.11 public void waitForWriteClose()

This function intentionally blocks until all write actions of the editor are finished.

6.12 private void insertText(String front, String back, String... optional)

This technique inserts in the last text field of a Def that had focus the String denoted by “front” at the last known cursor position or at the begin of the last known selection (if there was one), the String denoted by “back” at the last known cursor position (but after “front”) or at the end of the last known selection (if there was one). The String in the array “optional” are placed between “front” and “back” if there was no selection.

6.13 public void saveCurrentEntry()

This function saves the current displayed entry to the definitions, and is only called via *tonActionListener* (see 7.10) by the “Save entry” button. Also updates the database panel with the new entry.

6.14 public void updateDefinitions(Hashtable<String, WordDef> defs)

When the definitions are read by the Advisor class, a new library is created. To keep both programs synchronized, the definitions can be updated in this method.

6.15 CaretListener defCaretListener

Creates a new *CaretListener* object. Only overrides the *caretUpdate* function from the *CaretListener* class. This function keeps track of the last selected text field of the *Defs*, including the position of the cursor and selections. This is needed for inserting text using the layout buttons.

6.16 WindowAdapter windowClosed

Creates a new *WindowAdapter* object. Only overrides the *windowClosing* function from the *WindowListener* class. This function is called when the editor is closed and waits with closing till all writers are done (see 6.11) and notify the Advisor class that the editor was closed. Note that the editor does not really close, it only is not visible.

6.17 public void setVisible(Boolean visible)

Sets the visibility of the main frame.

6.18 DocumentListener textAreaDocListener

Creates a new *DocumentListener* object which regulates a default text for an text area. Overrides the following functions from the *DocumentListener* class:

6.18.1 changeUpdate(DocumentEvent e)

This method is called when any change occurs. No action is taken.

6.18.2 insertUpdate(DocumentEvent e)

This method is called when one or more characters are inserted. This function checks if this was the first character entered by the user. If so, remove all default text except for all characters which were inputted by the user. Also colors the text black in that case. Here is assumed that

the text is inserted at the beginning of the text area (is ensured by *CaretListener* 7.19). This function temporarily removes itself from the search bar and is executed on another thread because it manipulates data which will otherwise lead to infinite recursive calls to itself.

6.18.3 removeUpdate(DocumentEvent e)

This function is called when one or more characters are deleted. This function checks if there are still characters left. If none left, put the default text back and colour it grey. This function temporarily removes itself from the caller and is executed on another thread because it manipulates data which will otherwise lead to infinite recursive calls to itself.

6.19 CaretListener textAreaCaretListener

Creates a new *CaretListener* object. Only overrides the *caretUpdate* function from the *CaretListener* class. This function sets the position of the cursor to the beginning of the text area of the search bar. This function temporarily removes itself from the search bar and is executed on another thread because it manipulates data which will otherwise lead to infinite recursive calls to itself.

6.20 public void search()

Shows only those entries in the database panel that match the search requirements.

6.21 Action nullAction

Creates an action that does nothing. This action is used to disable the enter key for JTextAreas.

11.1.7 building.editor.Sort

This is a static class that can sort text, which is used for sorting the *WordDefs* in the database panel. It has the following functions:

7.0 Constructor

Deprecated function

This is a static class, so no instances of this class should be made.

7.1 public static Object[] textBubbleSort(String[] inArray, Object[] meta, int typeLetters, int typeNum, int sortingHint, int lengthHint)

Uses bubble sort to sort the *Strings* in “inArray” with corresponding metadata “meta”, using the typeLetters, typeNum, sortingHint and lengthHint as ruleset. The variable “meta” is optional. If “meta” is not given or null, then the *String* array is returned in order. If “meta” is not equal to null, then its length must be equal to “inArray”, and the returned value is the metadata in sorted order (using the strings for sorting). The comparison is done by the function *inOrder* (see 7.2).

7.2 public static boolean inOrder (String str1, Str2, int typeLetters, int typeNum, int sortingHint, int lengthHint)

Returns true if str1 >= str2, using typeLetters, typeNum, sortingHint and lengthHint as ruleset. Returns false otherwise.

11.1.8building.editor.Def

This class extends *JPanel* and contains the components for editing one definition of a word. It contains three *JComboBoxes* (dropdown menus), one *JTextArea* (text input) and one *JCheckBox* (checkbox). The leftmost *JComboBox* denotes whether the definition denotes a word definition or a NL-SfB class. If the first *JComboBox* denotes a word definition, then the second *JComboBox* denotes a definition type, and if the first *JComboBox* denotes a NL-SfB class, then the second *JComboBox* denotes either a NL-SfB class definition or a NL-SfB class specification. To also store the previous choice of the user when the other option is selected in the first *JComboBox*, two *JComboBoxes* are used, but only either one of them is displayed. The *JTextArea* denotes the definition and the *JComboBox* on the rightmost side is used for (de-)selecting the definition. The class has the following functions:

8.0 Constructor

The constructor can take the following parameters as input:

- **int type**
This denotes either the word definition or a NL-SfB class. This parameter is optional, but linked with “defValue” and “text”.
- **int defValue**
In the case that “type” denotes a word definition, this denotes a the definition type of a word, and in case that “type” denotes a NL-SfB class, this denotes the NL-SfB-class definition or specification. This parameter is optional, but linked with “type” and “text”.
- **String text**
This denotes the text that should be in the text area of the definition. This parameter is optional, but linked with “type” and “defValue”.
- **int width**
This denotes the width of the object. The height is set automatically. This parameter is optional.
- **int spacing**
This denotes the spacing used between the components.

8.1 public JComboBox<String> getTypeSelector()

Deprecated function

Returns the NL-SfB class/word definition selector. The returned object can only be used as READ-ONLY since there is no checking done. However, this allows to add listeners to this selector and makes it possible to use it in comparisons.

8.2 public JComboBox<String> getDefValueSelector()

Deprecated function

Returns the NL-SfB class definition/specification selector. The returned object can only be used as READ-ONLY since there is no checking done. However, this allows to add listeners to this selector and makes it possible to use it in comparisons.

8.3 public JComboBox<String> getDefDescrSelector()

Deprecated function

Returns the definition type selector. The returned object can only be used as READ-ONLY since there is no checking done. However, this allows to add listeners to this selector and makes it possible to use it in comparisons.

8.3 public JTextArea getDescrField()

Returns the description field. This function is not deprecated because the layout of this *JTextArea* is managed by the layout manager of a *JScrollPane*.

8.4 public boolean isSelected()

Returns true if the checkbox is selected. False otherwise.

8.5 public void setType(int i)

Sets the word definition or a NL-SfB class selector, where i is a number from one of the variables of the class *building.WordDef* (see 5.5.4).

8.6 public setDefValue(int i)

Sets the NL-SfB class definition or specification selector, where i is a number from one of the variables of the class *building.WordDef* (see 5.5.4).

8.7 setSpacing(int spacing)

Sets the spacing between the *Objects* in the *Def*.

8.8 public void setBounds(int x, int y, int width, int height)

Resizes and sets the location of the *Objects* in the *Def* such that they exactly fit in the resized *JPanel*.

8.9 protected void paintBorder(Graphics g)

Used to add a custom border to the *Def*.

8.10 ItemListener itemListener

Creates a new *ItemListener* object. Only overrides the *itemStateChanged* function from the *ItemListener* class. This listener checks if the first *JComboBox* changes from the option word definition to a NL-SfB class or vice versa. If such a change occurs, then set the second *JComboBox* accordingly.

11.1.9 building.editor.WordCell

This class extends *JPanel* and is a simple selectable cell containing text and supports showing focus and showing selection. Should be added to a *building.editor.WordCellGroup* (see 5.5.10) for easy handling interaction between multiple *WordCells*. The *WordCells* are used to display the entries in the database panel. This class has the following methods

9.0 Constructor

The constructor has as input a *String* containing the text to be displayed on the box.

9.1 public void setSelected(Boolean select)

Deprecated function

This function changes the look of the cell to a selected cell. This function is deprecated because setting cell selections should be done by a *building.editor.WordCellGroup* (see 5.5.10).

9.2 public void setFocus(Boolean select)

Deprecated function

This function changes the look of the cell to a cell that has focus . This function is deprecated because setting cell focus should be done by a *building.editor.WordCellGroup* (see 5.5.10).

9.3 public boolean isSelected()

Returns true if the cell is displayed as a selected cell. Returns false otherwise.

9.4 public boolean hasFocus()

Returns true if the cell is displayed as a focussed cell. Returns false otherwise. Note that this function does NOT determine whether the actual component has focus. It only tells you what it is currently displaying.

9.5 public String getText()

Returns the text that was set in the constructor (see 9.0).

9.6 protected void paintComponent(Graphics G)

Paints the background and the text of the cell.

9.6 protected void paintBorder(Graphics G)

Paints the custom borders for the cell.

11.1.10 building.editor.WordCellGroup

This class makes the cooperation between *building.editor.WordCells* (see 5.5.9) much easier. This class has the following methods:

10.0 Constructor

The constructor has two parameters as input:

- **int type**
This denotes the rules that the *WordCells* in the group have to follow. Must be one of the predefined values in this class. This parameter is optional.
- **WordCell... wcs**
This denotes an array of *WordCells* that should initially be added to the group. This parameter is optional.

11.1 MouseAdapter oneSelectMouseListener

Creates a new *MouseAdapter* object. Only overrides the *mousePressed* function from the *MouseListener* class. This function is used to (de-)select a *WordCell* that had a *ClickEvent* and set (visible) focus to that cell. Deselects any other *WordCell*. This *MouseAdapter* is used for the type `TYPE_SELECT_ONE_ONLY`.

11.2 public void setType(int type)

Set the rule for the *WordCells* in this group. Must be one of the predefined values in this class.

11.3 private void addRightListener(WordCell wc)

Adds the correct listener to a *WordCell*. Currently only the types `TYPE_NONE` and `TYPE_SELECT_ONE_ONLY` are available.

11.4 public void add(WordCell wc)

Adds the *WordCell* *wc* to the group. Note that you can add the same *WordCell* multiple times. However, this will probably result in strange behaviour.

11.5 public WordCell remove(WordCell wc)

Removes the *WordCell* *wc* from the list. Note that if the *WordCell* occurs more than once, only one of them is removed.

11.6 public void clear()

Removes all *WordCells* from the group.

11.7 public int size()

Returns the size of the group.

11.8 public ArrayList<WordCell> getGroupArrayList()

Returns a copy of the *WordCells* in this group

11.9 public WordCell getLastSelected()

Returns the last selected *WordCell*.

11.10 public void sortWordCells()

Uses the function *textBubbleSort* (see 7.1) *building.editor.Sort* class (see 5.5.7) to sort the *WordCells*.

11.11 public boolean contains(WordCell wc)

Returns true if the *WordCell* *wc* is in this group. Otherwise return false.

11.12 public boolean containsName(String name)

Returns true if the name occurs as text for at least one of the *WordCells* in this group. Otherwise return false.

11.2 Appendix A: building.BufferedReaderPlus

```
1
2  /* * * * * * * * * * *
3   * Created by MASdude   *
4   *   Last modified:    *
5   *   19-10-2017        *
6   *   (dd-mm-yyyy)      *
7   * * * * * * * * * */
8
9  package building;
10
11  // java packages
12  import java.io.BufferedReader;
13  import java.io.File;
14  import java.io.FileReader;
15  import java.io.IOException;
16  import java.io.Reader;
17
18  import java.lang.reflect.Array;
19
20  import java.util.ArrayList;
21  import java.util.List;
22
23
24  public class BufferedReaderPlus extends BufferedReader {
25      final private static String fileName = System.getProperty("user.dir") +
26          "\\building\\definitions.csv";
27
28      // Current linecounter in the file
29      private int lineCounter = 0;
30      // true iff in a multipleLineComment block
31      private boolean mutipleLineComment = false;
32
33      // Temporary storage for the mark option
34      private int markedLineCounter = 0;
35      private boolean markedMutipleLineComment = false;
36      private String markedBufferedLine = "";
37
38      // Stores the current line when reading cells (csv filetypes only)
39      private String bufferedLine = "";
40
41      // Stores the current comment String
42      private String commentString = "";
43
44      /* Denotes which type of comment is used in the file
45       * - -1 : no comments.
46       * - 0 : all text after '#' is comments.
47       * - 1 : all text after '/' is comments and
48       *       all text between '/*' and '*' /' is comments.
49       * - 2 : all text after '/*:\'
50       * - 3 : Own definition of single line comment
51       * Default is 0.
52       */
53      private int comment = 0;
54
55      // Boolean that denotes whether the input file is a csv or not
56      // Default is false
57      private boolean isCsv = false;
58
59      /*
60       * Constructors
61       */
62      public BufferedReaderPlus(Reader reader) {
63          super(reader);
64      }
65
66      public BufferedReaderPlus(Reader reader, int commentType) {
67          super(reader);
68          comment = commentType;
69
70          if (commentType == 0) {
71              commentString = "#";
72          }
73      }
74  }
```

```

73         } else if (commentType == 2) {
74             commentString = ":\\";
75         }
76     }
77
78     public BufferedReaderPlus(Reader reader, boolean isCsv) {
79         super(reader);
80         this.isCsv = isCsv;
81     }
82
83     public BufferedReaderPlus(Reader reader, int commentType, boolean isCsv) {
84         super(reader);
85         comment = commentType;
86         this.isCsv = isCsv;
87
88         if (commentType == 0) {
89             commentString = "#";
90
91         } else if (commentType == 2) {
92             commentString = ":\\";
93         }
94     }
95
96     public BufferedReaderPlus(Reader reader, String singleCommentType, boolean
isCsv) {
97         super(reader);
98         comment = 3;
99         this.isCsv = isCsv;
100
101         commentString = singleCommentType;
102     }
103
104     /*
105     * Reads a line and increases the lineCounter
106     */
107     @Override
108     public String readLine() throws IOException {
109         lineCounter++;
110         return super.readLine();
111     }
112
113     /*
114     * Reads a line while skipping empty lines and ignoring comments (characters
after '//').
115     * Also handles multiple line comment ('/*' ... '*' /' (without space)).
116     * Also removes beginning or trailing white spaces.
117     *
118     * Also removes the last two semi-colons (if present) iff the flag isCsv == true
119     */
120     public String readProcessedLine() throws IOException {
121         String line = "";
122
123         //
124         -----
125         if (comment == 0 || comment == 2 || comment == 3) { // single line comment
126             for (int pointer = 0; pointer < line.length() || line.length() == 0;
127                 pointer++) {
128                 while (line.length() == 0) {
129                     // If the previous line has a length of 0, read the next line,
130                     // increase the line counter
131                     // and reset the pointer
132                     line = this.readLine();
133                     pointer = 0;
134
135                     // If EOF reached, return null
136                     if (line == null) {
137                         return null;
138                     }
139
140                     if (isCsv) {
141                         // remove trailing semi-colons and trim the line
142                         line = removeCSVTrailings(line.trim()).trim();

```

```

140     }
141 }
142
143 // Checks the line for comments.
144 String checkLine = (commentString.length() + pointer <=
line.length() ? "" : null);
145 for (int i = 0; checkLine != null && i < commentString.length();
i++) {
146     checkLine += line.charAt(pointer + i);
147 }
148
149 if (checkLine != null && checkLine.equals(commentString)) {
150     // If there are comments, skip the last part of the line.\\
151     line = line.substring(0, pointer);
152 }
153 }
154
155 // Return the line
156 return line.trim();
157
158 //
-----
159 } else if (comment == 1) { // //, /* and */ comment
160     int startComment = 0; // Start location in the case of multiple line
comment.
161
162     for (int pointer = 0; pointer < line.length() || line.length() == 0;
pointer++) {
163         while (line.length() == 0) {
164             // If the previous line has a length of 0, read the next line,
increase the line counter
165             // and reset the pointer
166             line = this.readLine();
167             pointer = 0;
168
169             // If EOF reached, return null
170             if (line == null) {
171                 return null;
172             }
173
174             if (isCsv) {
175                 // remove trailing semi-colons and trim the line
176                 line = removeCSVTrailings(line.trim()).trim();
177             }
178         }
179
180         // Checks the line for comments
181         if (multipleLineComment) { // If there are multiple line comments
182             if (line.length() - 1 > pointer) {
183                 if (line.charAt(pointer) == '*' && line.charAt(pointer+1) ==
'/' ) {
184
185                     multipleLineComment = false; // Update the multiple line
comment flag.
186                     pointer += 2;
187
188                     // Remove the comment from the current line
189                     if (startComment != 0 && pointer != line.length()) {
190                         line = line.substring(0, startComment) +
line.substring(pointer, line.length());
191
192                     } else if (startComment != 0) {
193                         line = line.substring(0, startComment);
194
195                     } else if (pointer != line.length()) {
196                         line = line.substring(pointer, line.length());
197
198                     } else {
199                         line = "";
200                     }
201
202                     startComment = 0; // Reset the beginning of the comments.

```

```

203         pointer = -1;
204     }
205
206     } else if (line.length() - 1 == pointer) {
207         if (startComment != 0) {
208             line = line.substring(0, startComment);
209
210             } else {
211                 line = "";
212             }
213
214     }
215
216     } else { // If there is no multiple line comments
217         if (line.length() - 1 >= pointer &&
218             line.charAt(pointer) == '/' && line.charAt(pointer+1) ==
219             '/') {
220
221             // If there are comments, skip the last part of the line.
222             line = line.substring(0, pointer);
223
224             } else if (line.length() - 1 > pointer &&
225                 line.charAt(pointer) == '/' && line.charAt(pointer+1) ==
226                 '*') {
227
228                 startComment = pointer; // Remember the start of a comment.
229                 mutipleLineComment = true; // Set the multiple line comment
230                 flag.
231
232                 if (line.length() - pointer == 2) {
233                     line = line.substring(0, pointer);
234                 }
235
236                 pointer += 2; // Skip the two comment characters.
237             }
238         } // end for
239
240         // Return the line
241         return line.trim();
242
243         //
244         -----
245     } else { // No comment
246
247         while (line.length() == 0) {
248             // If the previous line has a length of 0, read the next line,
249             // increase the line counter
250             // and reset the pointer
251             line = this.readLine();
252
253             // If EOF reached, return null
254             if (line == null) {
255                 return null;
256             }
257
258             if (isCsv) {
259                 // remove trailing semi-colons and trim the line
260                 line = removeCSVTrailings(line.trim()).trim();
261             }
262
263             return this.readLine();
264         }
265     }
266
267     /*
268     * Removes trailing semi-colons from csv-files
269     */

```



```

270 public static String removeCSVTrailings(String line) {
271     int i = line.length() - 1;
272     while (i >= 0 && line.length() >= 1 && line.charAt(i) == ';') {
273         i--;
274     }
275
276     return line.substring(0, i + 1);
277 }
278
279 /*
280  * Reads a processed cell from a csv file.
281  *
282  * returns the next cell from the csv file.
283  * returns null iff EOF reached
284  * or if lineBlock == true and EOL reached.
285  *
286  * Uses processed lines iff processed == true.
287  * If no line was buffered, pick the next one iff lineBlock == false.
288  *
289  * csv-filetypes only!
290  */
291 public String readCSVCell(boolean processed) throws IOException {
292     return readCSVCell(processed, false);
293 }
294 public String readCSVCell(boolean processed, boolean lineBlock) throws
IOException {
295     if (!isCsv) throw new IllegalArgumentException("File type is not declared as
296         \".csv\"");
297
298     // Read the buffer
299     String line = bufferedLine;
300
301     // Checks the buffer length.
302     if (line.length() == 0) {
303         // If the bufferlength == 0, read new line iff allowed. Otherwise return
304         null.
305         if (lineBlock) return null;
306
307         // Reads the next line
308         line = processed ? readProcessedLine() : readLine();
309
310         // Check for the null-value
311         if (line == null) return null;
312     }
313
314     // Takes the substring of the beginning of the line till the first
315     semi-colon it finds.
316     for (int i = 0; i < line.length(); i++) {
317         if (line.charAt(i) == ';') {
318             bufferedLine = line.substring(i + 1);
319             return line.substring(0, i).trim();
320         }
321     }
322
323     // If no semi-colon is in the line, return the full line and set
324     bufferedLine to "".
325     bufferedLine = "";
326     return line;
327 }
328
329 /*
330  * Reads either all remaining processed cells on the buffered line
331  * or all cells on a new line from a csv file.
332  *
333  * Returns the data of each cell in an array in increasing order.
334  * returns null iff EOF reached
335  * or if lineBlock == true and EOL reached.
336  *
337  * Uses processed lines iff processed == true.
338  * If no line was buffered, pick the next one iff lineBlock == false.
339  *
340  * csv-filetypes only!
341  */

```

```

338     public ArrayList<String> readCSVLine(boolean processed) throws IOException {
339         return readCSVLine(processed, false);
340     }
341     public ArrayList<String> readCSVLine(boolean processed, boolean lineBlock)
342     throws IOException {
343         if (!isCsv) throw new IllegalArgumentException("File type is not declared as
344             \".csv\"");
345
346         // Read and clear the buffer
347         String line = bufferedLine;
348         bufferedLine = "";
349
350         // Checks the buffer length.
351         if (line.length() == 0) {
352             // If the bufferlength == 0, read new line iff allowed. Otherwise return
353             null.
354             if (lineBlock) return null;
355
356             // Reads the next line
357             line = processed ? readProcessedLine() : readLine();
358
359             // Check for the null-value
360             if (line == null) return null;
361         }
362
363         ArrayList<String> cells = new ArrayList<String>();
364         int prevColon = 0;
365
366         // Puts all Strings separated by a semi-colon in the ArrayList.
367         // If no semi-colon is present, return the full line as
368         // one element in the ArrayList
369         for (int i = 0; i <= line.length(); i++) {
370             if (i == line.length() ||
371                 line.charAt(i) == ';' ) {
372
373                 if (processed) {
374                     cells.add(line.substring(prevColon, i).trim());
375                 } else {
376                     cells.add(line.substring(prevColon, i));
377                 }
378                 prevColon = i + 1;
379             }
380         }
381
382         return cells;
383     }
384
385     /*
386     * Marks the present location in the stream.
387     * After reading 'readAheadLimit' characters, attempting to reset the stream may
388     fail.
389     */
390     @Override
391     public void mark(int readAheadLimit) throws IOException {
392         markedLineCounter = lineCounter;
393         markedMutipleLineComment = mutipleLineComment;
394         markedBufferedLine = bufferedLine;
395
396         super.mark(readAheadLimit);
397     }
398
399     /*
400     * Resets to the last marked point in the stream.
401     */
402     @Override
403     public void reset() throws IOException {
404         lineCounter = markedLineCounter;
405         mutipleLineComment = markedMutipleLineComment;
406         bufferedLine = markedBufferedLine;
407
408         super.reset();
409     }

```

```

407     /*
408     * Returns the current linecounter.
409     */
410     public int getLineCounter() {
411         return lineCounter;
412     }
413
414
415     /* -----
416     *      !!! WARNING !!!
417     *      SHOULD NOT BE USED
418     * -----
419     */
420     @Override
421     @Deprecated
422     public int read() {
423         throw new UnsupportedOperationException("Operation was not supported");
424     }
425
426     /* -----
427     *      !!! WARNING !!!
428     *      SHOULD NOT BE USED
429     * -----
430     */
431     @Override
432     @Deprecated
433     public int read(char[] cbuf, int off, int len) throws IOException {
434         throw new UnsupportedOperationException("Operation was not supported");
435     }
436
437     /*
438     * Converts any ArrayList to an array
439     *
440     * Input:
441     * - list : the input list
442     * - classValue : the input/output class type
443     *
444     * returns the elements from the output array in
445     * the same order as in the input List.
446     * returns null iff the given list is null
447     * or the given class == null
448     */
449     @SuppressWarnings("unchecked")
450     public static <A, B extends A> A[] listToArray(List<B> list, Class<B>
classValue) {
451         if (list == null) {
452             return null;
453         } else {
454             A[] array = (A[]) Array.newInstance(classValue, list.size());
455
456             for (int i = 0; i < list.size(); i++) {
457                 array[i] = list.get(i);
458             }
459
460             return (A[]) array;
461         }
462     }
463
464
465     /*
466     * Converts any array to an ArrayList
467     *
468     * Input:
469     * - array : the input array
470     * - classValue : the input/output class type
471     *
472     * returns the elements from the output ArrayList in
473     * the same order as in the input array.
474     * returns null iff the given list is null
475     * or the given class == null
476     */
477     @SuppressWarnings("unchecked")
478     public static <A, B extends A> ArrayList<A> arrayToArrayList(B[] array, Class<B>

```

```

classValue) {
479     if (array == null) {
480         return null;
481     }
482     else {
483         List<A> list = new ArrayList<A>(array.length);
484
485         for (int i = 0; i < array.length; i++) {
486             list.add((A) array[i]);
487         }
488
489         return (ArrayList<A>) list;
490     }
491 }
492
493
494
/*
-----
495  * To initiate the program
496  *
-----
*/
497
498 public static void main(String[] args) {
499     new Advisor();
500 }
501
502
503 public static void main(String[] args) {
504     try {
505         BufferedReaderPlus brp = new BufferedReaderPlus(new FileReader(new
File(fileName)), 1, true);
506
507         ArrayList<String> lineList = new ArrayList<String>();
508         String cell;
509
510         ArrayList<Object> tmp2 = new ArrayList<Object>();
511         tmp2.add("a");
512         Object[] tmp = listToArray(tmp2, Object.class);
513         System.out.println(tmp[0]);
514         // true/false:
515         // { line/cell , processed(T/F) , lineBlock(T/F) }
516         boolean[] choice = {false, true, true};
517
518         if (choice[0]) {
519             while ((lineList = brp.readCSVLine(choice[1], choice[2])) != null) {
520                 System.out.println(lineList);
521             }
522
523         } else {
524             while ((cell = brp.readCSVCell(choice[1], choice[2])) != null) {
525                 System.out.println(cell);
526             }
527         }
528
529         brp.close();
530     } catch (IOException e){
531         System.out.println(e);
532     }/**/*
533 }*/
534 }
535
536
537
538

```

11.3 Appendix B: building.log.Log

```
1  /* * * * * * * * * * *
2  * Created by MASdude *
3  * Last modified: *
4  * 19-10-2017 *
5  * (dd-mm-yyyy) *
6  * * * * * * * * * */
7
8
9  package building.log;
10
11  // java packages
12  import java.io.BufferedWriter;
13  import java.io.FileWriter;
14  import java.io.IOException;
15  import java.io.PrintWriter;
16
17  import java.text.DateFormat;
18  import java.text.SimpleDateFormat;
19
20  import java.util.Arrays;
21  import java.util.Date;
22
23
24  public class Log {
25      final private static String logFile = System.getProperty("user.dir") +
26          "\\building\\log\\log.txt";
27      final private static String ls = System.getProperty("line.separator");
28      private static PrintWriter writer;
29
30      public static void write(boolean text) {
31          write(text, true);
32      }
33      public static void write(boolean text, boolean showDate) {
34          if (text) {
35              write("true", showDate);
36          } else {
37              write("false", showDate);
38          }
39      }
40
41      public static void write(char text) {
42          write(text, true);
43      }
44      public static void write(char text, boolean showDate) {
45          write(text + "", showDate);
46      }
47
48      public static void write(int number) {
49          write(number, true);
50      }
51      public static void write(int number, boolean showDate) {
52          write(Integer.toString(number), showDate);
53      }
54
55      public static void write(double number) {
56          write(number, true);
57      }
58      public static void write(double number, boolean showDate) {
59          write(Double.toString(number), showDate);
60      }
61
62      public static void write(Exception e) {
63          write(e, true, true);
64      }
65      public static void write(Exception e, boolean full) {
66          write(e, full, true);
67      }
68      public static void write(Exception e, boolean full, boolean showDate) {
69          if (full) {
70              String[] text = Arrays.toString(e.getStackTrace()).split(", ");
71              String message = "[ERROR] " + e.getClass().getName() + ": " +
72                  e.getMessage();
73              write(message, showDate);
74          }
75      }
76  }
```

11.4 Appendix C: building.GenerateDesign

```
1
2  /* * * * * * * * * * *
3   * Created by MASdude   *
4   *   Last modified:    *
5   *   19-10-2017       *
6   *   (dd-mm-yyyy)     *
7   * * * * * * * * * */
8
9  package building;
10
11  // Own packages
12  import building.log.Log;
13
14  // Java packages
15  import java.io.FileNotFoundException;
16  import java.io.PrintWriter;
17
18  import java.util.ArrayList;
19  import java.util.Hashtable;
20
21
22  public class GenerateDesign {
23      private static ArrayList<WordDef> wordsSeen = new ArrayList<WordDef>();
24
25      protected static void generateDesign(Hashtable<String, WordDef> definitions,
26      String text, int mostWords) {
27          Log.write("[INFO]      Start design generation");
28
29          int warnings = 0;
30          boolean successfull = true;
31
32          // Create a writer to the outputfiles
33          try (PrintWriter pw = new PrintWriter(Advisor.outputFile)) {
34              try (PrintWriter pw2 = new PrintWriter(Advisor.outputFile2)) {
35                  try (PrintWriter pw3 = new PrintWriter(Advisor.outputFile3)) {
36
37                      // If empty input, log a warning and break. Otherwise generate
38                      // the design.
39                      if (text.equals("")) {
40                          pw.println("No input information was available");
41                          Log.write("[WARNING] Input was empty");
42
43                      } else {
44                          // Split the input into lines
45                          String[] lines = text.split("\\.");
46
47                          for (int type = WordDef.FIRST_TYPE; type <=
48                          WordDef.FINAL_TYPE; type++) {
49                              // Clears the list
50                              wordsSeen.clear();
51
52                              // Checks if the header should be printed
53                              boolean[] headerWritten = {false, false};
54
55                              for (int i = 0; i < lines.length; i++) {
56                                  // Split each line into separate words
57                                  String[] words = lines[i].split(" ");
58
59                                  // Iterates over the words for each line
60                                  for (int j = 0; j < words.length; j++) {
61                                      // Skip empty strings
62                                      if (words[j].length() == 0) {
63                                          continue;
64                                      }
65
66                                      // Search a definition of the word
67                                      String line = ""; // = words[j].toLowerCase();
68                                      WordDef[] defs = new WordDef[(j + mostWords <=
69                                      words.length ? mostWords : words.length - j)];
70                                      String[] defLines = new String[defs.length];
71
72                                      // The definition might be more then a single
73                                      word.
```

```

69         // Test if this is the case. One can stop with
70         checking when
71         // the current number of words is larger then
72         the longest
73         // occurring series of words in a row.
74         for (int k = 0; k < defs.length; k++) {
75             line += (k == 0 ? "" : " ") + words[j +
76                 k].toLowerCase();
77             defs[k] = definitions.get(line);
78
79             // Check if a definition was already used
80             before.
81             // If so, set it to null.
82             // Else add the word to the list.
83             if (defs[k] != null &&
84                 wordsSeen.contains(defs[k])) {
85                 defs[k] = null;
86             } else {
87                 wordsSeen.add(defs[k]);
88             }
89
90             defLines[k] = line;
91         }
92
93         // Debug
94         boolean allNull = true;
95         for (int k = 0; k < defs.length; k++) {
96             if (defs[k] != null) {
97                 allNull = false;
98                 break;
99             }
100         }
101         if (allNull) {
102             // If the word was not defined, log warning
103             and continue
104             Log.write("[WARNING] Could not find word in
105             definitions: \"\" + words[j] +
106             "\" [line: \"\" + (i + 1) + \"\", word:
107             \"\" + (j + 1) + \"]");
108
109             warnings++;
110             continue;
111         }
112
113         /*
114         -----
115         * Your code here
116         *
117         -----
118         */
119         for (int k = 0; k < defs.length; k++) {
120             if (defs[k] == null) continue;
121
122             // Output 1 & 2
123             ArrayList<String> typeDef =
124             defs[k].getDef(type);
125             PrintWriter pwUsed = (type == WordDef.OTHER
126                 ? pw : pw2);
127
128             if (typeDef.size() != 0) {
129                 // Write a header iff there occurs at
130                 least one element in it.
131                 if (type != WordDef.OTHER &&
132                     !headerWritten[0]) {
133                     headerWritten[0] = true;
134                     pw2.println("<b>" +
135                         WordDef.getTypeString(type) + "

```

```

125         :</b>");
126     }
127     pwUsed.print("<b>" + defLines[k] + "</b>"
128         : " ");
129     for (int typeDefLength = 0;
130         typeDefLength < typeDef.size();
131         typeDefLength++) {
132         if (typeDefLength == 0) {
133             pwUsed.print(typeDef.get(typeDefLength));
134         } else {
135             pwUsed.print(", " +
136                 typeDef.get(typeDefLength));
137         }
138         if (typeDefLength + 1 ==
139             typeDef.size()) {
140             pwUsed.println();
141         }
142     }
143     // End output 1 & 2
144     // Output 3
145     ArrayList<String> values =
146         defs[k].getValues(type);
147     if (values.size() != 0) {
148         // Write a header iff there occurs at
149         // least one element in it.
150         if (type != WordDef.OTHER &&
151             !headerWritten[1]) {
152             headerWritten[1] = true;
153             pw3.println("<b>" +
154                 WordDef.getTypeString(type) + "
155                 :</b>");
156         }
157         for (int valueLength = 0; valueLength <
158             values.size(); valueLength++) {
159             if (valueLength == 0) {
160                 pw3.print(values.get(valueLength))
161                 ;
162             } else {
163                 pw3.print(", " +
164                     values.get(valueLength));
165             }
166             if (valueLength + 1 ==
167                 values.size()) {
168                 pw3.println();
169             }
170         }
171         // End output 3
172     }
173     } // end for words
174 } // end for lines
175
176 if (type != WordDef.OTHER && headerWritten[0]) {
177     pw2.println();
178 }
179

```



```

180         if (type != WordDef.OTHER && headerWritten[1]) {
181             pw3.println();
182         }
183     } // endfor wordType
184
185     }
186
187     // If the file was not found or could not be opened
188     } catch (FileNotFoundException e) {
189         successfull = false;
190         Log.write("[ERROR] Could not find/create file: " +
191             Advisor.outputFile3.getPath());
192         //e.printStackTrace();
193     }
194     } catch (FileNotFoundException e) {
195         successfull = false;
196         Log.write("[ERROR] Could not find/create file: " +
197             Advisor.outputFile2.getPath());
198         //e.printStackTrace();
199     }
200     } catch (FileNotFoundException e) {
201         successfull = false;
202         Log.write("[ERROR] Could not find/create file: " +
203             Advisor.outputFile.getPath());
204         //e.printStackTrace();
205     }
206
207     // Log the final result
208     if (successfull) {
209         if (warnings == 0) {
210             Log.write("[INFO] Design was generated successfully with no
211                 warnings");
212
213         } else if (warnings == 1) {
214             Log.write("[INFO] Design was generated with 1 warning");
215
216         } else {
217             Log.write("[INFO] Design was generated with " + warnings + "
218                 warnings");
219         }
220     }
221
222     }
223
224     /*
225     -----
226     * To initiate the program
227     *
228     -----
229     */
230     public static void main(String[] args) {
231         new Advisor();
232     }
233 }

```

11.5 Appendix D: building.WordDef

```
1
2  /* * * * * *
3   * Created by MASdude   *
4   *   Last modified:    *
5   *   19-10-2017       *
6   *   (dd-mm-yyyy)     *
7   * * * * * */
8
9  package building;
10
11  // Java packages
12  import java.util.ArrayList;
13
14
15  public class WordDef {
16      final public static int OTHER = 0;
17      final public static int GROUND_SUBSTRUCTURE = 1;
18      final public static int STRUCTURE_PRIMARY_ELEMENTS = 2;
19      final public static int SECONDARY_ELEMENTS = 3;
20      final public static int FINISHING = 4;
21      final public static int SERVICES_MAINLY_MECHANICAL = 5;
22      final public static int SERVICES_MAINLY_ELECTRICAL = 6;
23      final public static int FACILITIES = 7;
24      final public static int FITTINGS = 8;
25      final public static int TERRAIN = 9;
26      final public static int DEMOLITION = 10;
27
28      final public static int NONE = -1;
29      final public static int FIRST_TYPE = 0;
30      final public static int FINAL_TYPE = 10;
31      final public static int TOTAL_TYPES = 11;
32
33      final public static int VALUE = 1;
34      final public static int DEF = 0;
35
36      private ArrayList<String>[] defs;
37      private ArrayList<String>[] values;
38
39      /*
40       * Constructor
41       */
42      @SuppressWarnings("unchecked") // Used to create an array of ArrayLists
43      public WordDef(ArrayList<String>[] newDefs, ArrayList<String>[] newValues) {
44          if (newDefs == null || newDefs.length != TOTAL_TYPES)
45              throw new IllegalArgumentException("Incorrect length for given
46              definition.");
47
48          if (newValues == null || newValues.length != TOTAL_TYPES)
49              throw new IllegalArgumentException("Incorrect length given definition.");
50
51          defs = (ArrayList<String>[]) new ArrayList[TOTAL_TYPES];
52          for (int i = FIRST_TYPE; i <= FINAL_TYPE; i++) {
53              defs[i] = newDefs[i];
54          }
55
56          values = (ArrayList<String>[]) new ArrayList[TOTAL_TYPES];
57          for (int i = FIRST_TYPE; i <= FINAL_TYPE; i++) {
58              values[i] = newValues[i];
59          }
60
61      /*
62       * Sets the definition of the word, regarding the given type.
63       *
64       * SHOULD NOT BE USED!
65       * Create a new wordDef instead.
66       */
67      @Deprecated
68      private void setDef(ArrayList<String> newDefs, int i) {
69          defs[i] = newDefs;
70      }
71
72      /*
```

11.6 Appendix E: building.Advisor

```
73      * Returns the definition of the word, regarding the given type.
74      */
75      public ArrayList<String> getDef(int i) {
76          return (ArrayList<String>) defs[i];
77      }
78
79      /*
80      * Returns the definitions of the word of all types
81      */
82      public ArrayList<String>[] getAllDef() {
83          return defs;
84      }
85      /*
86      * Returns the value of the word, regarding the given types.
87      */
88      public ArrayList<String> getValues(int i) {
89          return (ArrayList<String>) values[i];
90      }
91
92      /*
93      * Returns the name of the type as string.
94      */
95      public static String getTypeString(int num) {
96          if (num == OTHER) {
97              return "DEFINITION";//"OTHER";
98
99          } else if (num == GROUND_SUBSTRUCTURE) {
100              return "10-GROUND, SUBSTRUCTURE";
101
102          } else if (num == STRUCTURE_PRIMARY_ELEMENTS) {
103              return "20-STRUCTURE PRIMARY ELEMENTS, SKELETON";
104
105          } else if (num == SECONDARY_ELEMENTS) {
106              return "30-SECONDARY ELEMENTS, OPENINGS";
107
108          } else if (num == FINISHING) {
109              return "40-FINISHING";
110
111          } else if (num == SERVICES_MAINLY_MECHANICAL) {
112              return "50-SERVICES MAINLY MECHANICAL";
113
114          } else if (num == SERVICES_MAINLY_ELECTRICAL) {
115              return "60-SERVICES MAINLY ELECTRICAL";
116
117          } else if (num == FACILITIES) {
118              return "70-FACILITIES";
119
120          } else if (num == FITTINGS) {
121              return "80-FITTINGS";
122
123          } else if (num == TERRAIN) {
124              return "90-TERRAIN";
125
126          } else if (num == DEMOLITION) {
127              return ("120-DEMOLITION");
128
129          } else if (num == NONE) {
130              return ("NONE");
131
132          } else {
133              return " - [TYPE NOT DEFINED] - ";
134          }
135      }
136
137      /*
138      * Returns the number represented by the string
139      * assuming that it is one of the def types.
140      * Returns NONE if not.
141      */
142      public static int getTypeFromString(String text) {
143          text = text.toUpperCase();
144
145          if (text.equals("OTHER") || text.equals("DEFINITION")) {
```

```

146         return OTHER;
147
148     } else if (text.equals("10-GROUND, SUBSTRUCTURE")) {
149         return GROUND_SUBSTRUCTURE;
150
151     } else if (text.equals("20-STRUCTURE PRIMARY ELEMENTS, SKELETON")) {
152         return STRUCTURE_PRIMARY_ELEMENTS;
153
154     } else if (text.equals("30-SECONDARY ELEMENTS, OPENINGS")) {
155         return SECONDARY_ELEMENTS;
156
157     } else if (text.equals("40-FINISHING")) {
158         return FINISHING;
159
160     } else if (text.equals("50-SERVICES MAINLY MECHANICAL")) {
161         return SERVICES_MAINLY_MECHANICAL;
162
163     } else if (text.equals("60-SERVICES MAINLY ELECTRICAL")) {
164         return SERVICES_MAINLY_ELECTRICAL;
165
166     } else if (text.equals("70-FACILITIES")) {
167         return FACILITIES;
168
169     } else if (text.equals("80-FITTINGS")) {
170         return FITTINGS;
171
172     } else if (text.equals("90-TERRAIN")) {
173         return TERRAIN;
174
175     } else if (text.equals("120-DEMOLITION")) {
176         return DEMOLITION;
177
178     } else {
179         return NONE;
180     }
181 }
182
183 /*
184  * Returns the name of def/value type of the word
185  */
186 public static String getDevValueString(int num) {
187     if (num == DEF) {
188         return "NL-SfB class definition";
189
190     } else if (num == VALUE) {
191         return "Specification";
192
193     } else {
194         return " - [TYPE NOT DEFINED] - ";
195     }
196 }
197
198 /*
199  * Returns the number represented by the string
200  * assuming that it is a def or a value type.
201  * Returns NONE if not.
202  */
203 public static int getDefValueFromString(String text) {
204     text = text.toUpperCase();
205
206     // The text DEF and VALUE are used for backward compatability of
207     // documents created by versions older than 2.0.1
208     if (text.equals("DEF") || text.equals("NL-SfB class".toUpperCase()) ||
209         text.equals("NL-SfB class definition".toUpperCase())) {
210         return DEF;
211
212     } else if (text.equals("VALUE") ||
213         text.equals("Specification".toUpperCase())) {
214         return VALUE;
215
216     } else {
217         return NONE;
218     }
219 }

```

```

217     }
218
219     @Override
220     public String toString() {
221         String text = this.getClass().getCanonicalName() + "[";
222
223         for (int i = FIRST_TYPE; i <= FINAL_TYPE; i++) {
224             text += getTypeString(i) + "={";
225
226             for (int j = 0; j < defs[i - FIRST_TYPE].size(); j++) {
227                 text += defs[i - FIRST_TYPE].get(j);
228
229                 text += (j + 1 != defs[i - FIRST_TYPE].size() ? ", " : "");
230             }
231
232             text += "}" + (i != FINAL_TYPE ? ", " : "");
233         }
234
235         return text + "]";
236     }
237
238     /*
239     -----
240     * To initiate the program
241     *
242     -----
243     */
244     public static void main(String[] args) {
245         new Advisor();
246     }

```

11.7 Appendix F: building.editor.DefFileEditor

```
1
2  /* * * * * * * * * * *
3   * Created by MASdude   *
4   *   Last modified:    *
5   *   19-10-2017       *
6   *   (dd-mm-yyyy)     *
7   * * * * * * * * * */
8
9  package building.editor;
10
11  // Own packages
12  import building.Advisor;
13  import building.BufferedReaderPlus;
14  import building.WordDef;
15  import building.log.Log;
16
17  // Java packages
18  import java.awt.Color;
19  import java.awt.Dimension;
20  import java.awt.Insets;
21  import java.awt.event.ActionEvent;
22  import java.awt.event.ActionListener;
23  import java.awt.event.ComponentAdapter;
24  import java.awt.event.ComponentEvent;
25  import java.awt.event.WindowAdapter;
26  import java.awt.event.WindowEvent;
27
28  // Check if all are used !!!
29  import java.io.BufferedInputStream;
30  import java.io.BufferedOutputStream;
31  import java.io.BufferedWriter;
32  import java.io.File;
33  import java.io.FileInputStream;
34  import java.io.FileOutputStream;
35  import java.io.FileWriter;
36  import java.io.IOException;
37
38  // Check if all are used !!!
39  import java.util.ArrayList;
40  import java.util.Enumeration;
41  import java.util.Hashtable;
42  import java.util.Iterator;
43  import java.util.Map;
44  import java.util.Map.Entry;
45  import java.util.Set;
46
47  import javax.swing.ImageIcon;
48  import javax.swing.JButton;
49  import javax.swing.JComboBox;
50  import javax.swing.JFrame;
51  import javax.swing.JOptionPane;
52  import javax.swing.JPanel;
53  import javax.swing.JScrollPane;
54  import javax.swing.JTextArea;
55  import javax.swing.SwingUtilities;
56  import javax.swing.event.CaretEvent;
57  import javax.swing.event.CaretListener;
58
59  // new
60  import javax.swing.JLabel;
61  import javax.swing.event.DocumentEvent;
62  import javax.swing.event.DocumentListener;
63  import javax.swing.Action;
64  import javax.swing.AbstractAction;
65  import javax.swing.KeyStroke;
66  import java.awt.event.KeyEvent;
67
68
69  public class DefFileEditor {
70      final private static String title = "Database Manager BOK";
71      final private static int spacing = 7;
72      final private static int TEXT_LAYOUT_HEIGHT = 30;
73      final private static int BUTTON_HEIGHT = 20;
```

```

74     final private static int WORD_CELL_HEIGHT = 30;
75     final private static int SCROLL_SPEED = 16;
76     final private static int DEF_HEIGHT = 30;
77     final private static int TEXT_HEIGHT = 16;
78     final private static int TEXT_SPACING = 4;
79
80     final private Advisor advisor;
81     private Hashtable<String, WordDef> definitions;
82     private Boolean canClose = true;
83
84     private ArrayList<Def> allDefs;
85     private WordCellGroup wcg;
86     private ArrayList<WordCell> allWordCells;
87     private File bacFile;
88     private File defFile;
89
90     private volatile JTextArea prevFocusedDefField = null;
91     private volatile Integer startSelection = 0;
92     private volatile Integer endSelection = 0;
93
94     private JFrame mainFrame;
95
96     // Layout
97     private JPanel textLayoutPanel;
98     private JButton bold;
99     private JButton italic;
100    private JButton underline;
101    private JButton link;
102    // OPTIONAL!!!
103    //private JButton color;
104
105    // Def
106    private JPanel wordPanel;
107    private JScrollPane wordScrollPane;
108    private JTextArea word;
109    private JScrollPane defScrollPane;
110    private JPanel defPanel;
111    private JButton addDef;
112
113    private JLabel tokenLabel;
114    private JLabel inputWordLabel;
115    private JLabel defLabel;
116
117    private JButton editEntry;
118    private JButton deleteEntry;
119    private JButton saveEntry;
120    private JButton deleteSelectedDefinitions;
121    private JButton saveChanges;
122
123    // Search
124    private JPanel searchPanel;
125    private JButton searchButton;
126    private JScrollPane searchScrollPane;
127    private JTextArea searchBar;
128    private JScrollPane searchResultsScrollPane;
129    private JPanel searchResultsPanel;
130
131    private boolean textTyped = false;
132    final private String searchBeginInputText = "Type something and press search to
search in the database";
133
134    private JLabel searchLabel;
135
136    /*
-----
137    * Constructor
138    *
-----
139    */
140    public DefFileEditor(Advisor advisor, File defFile, File bacFile,
141                        Hashtable<String, WordDef> defs, int locX, int locY) {

```



```

142     this.advisor = advisor;
143     this.defFile = defFile;
144     this.bacFile = bacFile;
145
146     synchronized(defs) {
147         definitions = defs;
148     }
149
150     createBACThread().start();
151
152     SwingUtilities.invokeLater(new Runnable() {
153         @Override
154         public void run() {
155             allDefs = new ArrayList<Def>();
156             allWordCells = new ArrayList<WordCell>();
157
158             mainFrame = new JFrame();
159             mainFrame.setLayout(null);
160             mainFrame.setTitle(title);
161             mainFrame.setLocation(locX, locY);
162             mainFrame.setSize(1000, 800);
163             mainFrame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
164
165             textLayoutPanel = new JPanel(null);
166             wordPanel = new JPanel(null);
167             searchPanel = new JPanel(null);
168
169             mainFrame.add(textLayoutPanel);
170             mainFrame.add(wordPanel);
171             mainFrame.add(searchPanel);
172
173             wcg = new WordCellGroup(WordCellGroup.TYPE_SELECT_ONE_ONLY);
174
175             // tmp
176             Color red = new Color(41, 56, 125); // TUE red
177             Color blue = new Color(147, 159, 219); // TUE blue
178             Color gray = new Color(150, 150, 150); // gray
179             textLayoutPanel.setBackground(red);
180             wordPanel.setBackground(blue);
181             searchPanel.setBackground(gray);
182             //searchPanel.setBackground(new Color(211, 211, 211));
183             mainFrame.getContentPane().setBackground(gray);
184
185             // Text layout panel
186             bold = new JButton("Bold");
187             italic = new JButton("Italic");
188             underline = new JButton("Underline");
189             link = new JButton("Link");
190
191             bold.setSize(60, BUTTON_HEIGHT);
192             italic.setSize(64, BUTTON_HEIGHT);
193             underline.setSize(90, BUTTON_HEIGHT);
194             link.setSize(60, BUTTON_HEIGHT);
195
196             int height = (TEXT_LAYOUT_HEIGHT - BUTTON_HEIGHT) / 2;
197             bold.setLocation(spacing, height);
198             italic.setLocation(bold.getWidth() + 2*spacing, height);
199             underline.setLocation(bold.getWidth() + italic.getWidth() +
200                                3*spacing, height);
201             link.setLocation(bold.getWidth() + italic.getWidth() +
202                             underline.getWidth() + 4*spacing, height);
203
204             textLayoutPanel.add(bold);
205             textLayoutPanel.add(italic);
206             textLayoutPanel.add(underline);
207             textLayoutPanel.add(link);
208
209             // Word panel
210             word = new JTextArea("");
211             wordScrollPane = new JScrollPane(word);
212             wordScrollPane.setSize(200, 20);

```



```

wordScrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLL
LLBAR_NEVER);
213
wordScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR
R_NEVER);
214
wordPanel.add(wordScrollPane);
215
defPanel = new JPanel(null);
216
defScrollPane = new JScrollPane(defPanel);
217
defScrollPane.setVerticalScrollBar().setUnitIncrement(SCROLL_SPEED);
218
219
defScrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLL
LLBAR_NEVER);
220
defScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR
_AS_NEEDED);
221
wordPanel.add(defScrollPane);
222
addDef = new JButton("Add definition");
223
defPanel.add(addDef);
224
225
226
227
tokenLabel = new JLabel("<html><font color='white'>Token
configuration:</font><html>");
228
inputWordLabel = new JLabel("<html><font color='white'>Input
word:</font><html>");
229
defLabel = new JLabel("<html><font color='white'>Assign
definition(s), classe(s) and specification(s):</font><html>");
230
231
tokenLabel.setSize(200, TEXT_HEIGHT);
232
tokenLabel.setLocation(spacing, TEXT_SPACING);
233
wordPanel.add(tokenLabel);
234
235
inputWordLabel.setSize(100, TEXT_HEIGHT);
236
wordPanel.add(inputWordLabel);
237
238
defLabel.setSize(500, TEXT_HEIGHT);
239
wordPanel.add(defLabel);
240
241
242
// Buttons
243
editEntry = new JButton("Edit token");
244
deleteEntry = new JButton("Delete token");
245
saveEntry = new JButton("Save token");
246
deleteSelectedDefinitions = new JButton("Del selected def(s)");
247
saveChanges = new JButton("Save changes");
248
249
wordPanel.add(editEntry);
250
wordPanel.add(deleteEntry);
251
wordPanel.add(saveEntry);
252
wordPanel.add(deleteSelectedDefinitions);
253
wordPanel.add(saveChanges);
254
255
256
// Search panel
257
searchButton = new JButton("Search");
258
searchButton.setSize(80, 20);
259
searchPanel.add(searchButton);
260
261
searchBar = new JTextArea(searchBeginInputText);
262
searchBar.setForeground(Color.GRAY);
263
264
searchScrollPane = new JScrollPane(searchBar);
265
266
searchScrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SC
ROLLBAR_NEVER);
267
searchScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLL
BAR_NEVER);
268
searchScrollPane.setSize(0, 20);
269
searchPanel.add(searchScrollPane);
270
searchResultsPanel = new JPanel(null);

```

```

271 searchResultsPanel.setLayout(null);
272 searchResultsScrollPane = new JScrollPane(searchResultsPanel);
273
274 searchResultsScrollPane.getVerticalScrollBar().setUnitIncrement(SCROLL_
    SPEED);
275
276 searchResultsScrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL
    NTAL_SCROLLBAR_NEVER);
277
278 searchResultsScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL
    _SCROLLBAR_AS_NEEDED);
279
280 searchPanel.add(searchResultsScrollPane);
281
282 searchLabel = new JLabel("Database queries:");
283 searchLabel.setSize(150, TEXT_HEIGHT);
284 searchLabel.setLocation(spacing, TEXT_SPACING);
285 searchPanel.add(searchLabel);
286
287 // Set the keybindings to the JTextAreas
288 searchBar.getInputMap().put(KeyStroke.getKeyStroke(KeyEvent.VK_ENTER,
    0), "EnterPressedSearch");
289 searchBar.getActionMap().put("EnterPressedSearch", new
    AbstractAction() {
290     @Override
291     public void actionPerformed(ActionEvent e) {
292         search();
293     }
294 });
295
296 word.getInputMap().put(KeyStroke.getKeyStroke(KeyEvent.VK_ENTER, 0),
    "null");
297 word.getActionMap().put("null", nullAction);
298
299 // Add the ActionListener to the buttons
300 addDef.addActionListener(buttonActionListener);
301 bold.addActionListener(buttonActionListener);
302 italic.addActionListener(buttonActionListener);
303 underline.addActionListener(buttonActionListener);
304 link.addActionListener(buttonActionListener);
305 editEntry.addActionListener(buttonActionListener);
306 deleteEntry.addActionListener(buttonActionListener);
307 saveEntry.addActionListener(buttonActionListener);
308 deleteSelectedDefinitions.addActionListener(buttonActionListener);
309 saveChanges.addActionListener(buttonActionListener);
310 searchButton.addActionListener(buttonActionListener);
311
312 // Add the WindowListener to the mainFrame
313 mainFrame.addWindowListener(windowClosed);
314
315 // Add documentListener and caretListener to the search bar
316 searchBar.getDocument().addDocumentListener(textAreaDocListener);
317 searchBar.addCaretListener(textAreaCaretListener);
318
319 // Add the component listener
320 mainFrame.addComponentListener(windowResizeListener);
321
322 // Set image
323 mainFrame.setIconImage(new
    ImageIcon(this.getClass().getClassLoader().getResource("building/tue.png")).getImage());
324
325 // Set the frame visible
326 mainFrame.setVisible(true);
327
328 // Update the GUI elements
329 windowResizeListener.componentResized(null);
330
331 synchronized(definitions) {
332     Enumeration<String> defEnum = definitions.keys();
333     while (defEnum.hasMoreElements()) {

```

```

333         addSearchResult(defEnum.nextElement());
334     }
335 }
336
337 wcg.sortWordCells();
338 updateGUI(true);
339 }
340 });
341 }
342
343 /*
-----
344 * Create a thread that makes a backup file from the definition file.
345 *
-----
346 */
347 Thread createBACThread() {
348     return new Thread("BAC thread") {
349         @Override
350         public void run() {
351             synchronized(canClose) {
352                 waitForWriteClose();
353                 canClose = false;
354
355                 try (BufferedInputStream bis = new BufferedInputStream(new
356                     FileInputStream(defFile));
357                     BufferedOutputStream bos = new BufferedOutputStream(new
358                         FileOutputStream(bacFile)))
359                 {
360                     // Read the def file and write to the bac file in data parts
361                     // of 1 kB to speed-up the process.
362                     byte[] data = new byte[1024];
363
364                     int len = data.length;
365                     while (len == data.length) {
366                         len = bis.read(data, 0, data.length);
367                         if (len <= 0) break;
368                         bos.write(data, 0, len);
369                     }
370                 } catch (IOException e) {
371                     Log.write(e);
372                     e.printStackTrace();
373                 }
374
375                 canClose = true;
376             }
377         }
378     };
379 }
380
381 /*
-----
382 * Adds a def to the defPanel
383 *
-----
384 */
385 private void addDef(Def def) {
386     allDefs.add(def);
387     defPanel.add(def);
388
389     def.getDescrField().getInputMap().put(KeyStroke.getKeyStroke(KeyEvent.VK_ENTER
390         , 0), "null");
391     def.getDescrField().getActionMap().put("null", nullAction);
392     def.revalidate();
393     def.getDescrField().addCaretListener(defCaretListener);
394 }
395
396 /*

```

```

-----
394  * Removes a def from the defPanel
395  *
-----

396  */
397  private void removeDef(Def def) {
398      defPanel.remove(def);
399      allDefs.remove(def);
400      def.getDescrField().removeCaretListener(defCaretListener);
401      if (prevFocusedDefField != null && prevFocusedDefField.equals(def)) {
402          prevFocusedDefField = null;
403      }
404
405      mainFrame.repaint();
406  }
407
408  /*
-----

409  * Clears all defs from the list
410  *
-----

411  */
412  private void clearDefs() {
413      for (int i = 0; i < allDefs.size(); i++) {
414          defPanel.remove(allDefs.get(i));
415      }
416
417      allDefs.clear();
418  }
419
420  /*
-----

421  * Adds a search result to the searchResultsPanel
422  *
-----

423  */
424  private void addSearchResult(String name) {
425      WordCell wc = new WordCell(name);
426
427      synchronized(wcg) {
428          synchronized(allWordCells) {
429              wcg.add(wc);
430              allWordCells.add(wc);
431              int size = wcg.size();
432              int wordCellWidth = searchResultsScrollPane.getWidth() / 2;
433              wc.setLocation((size/2 == size/2.0 ? wordCellWidth : 0),
434                          WORD_CELL_HEIGHT * ((wcg.size() - 1) / 2));
435              wc.setSize(wordCellWidth, WORD_CELL_HEIGHT);
436
437              searchResultsPanel.add(wc);
438          }
439      }
440  }
441
442  /*
-----

443  * Removes a seearch result from the searchResultsPanel
444  *
-----

445  */
446  private void removeSearchResult(WordCell wc) {
447      synchronized(wcg) {
448          synchronized(allWordCells) {
449              wcg.remove(wc);

```

```

450         allWordCells.remove(wc);
451         searchResultsPanel.remove(wc);
452     }
453 }
454
455
456 /*
457  * Removes a search result from the searchResultsPanel
458  */
459 private void hideSearchResult(WordCell wc) {
460     synchronized(wcg) {
461         wcg.remove(wc);
462         searchResultsPanel.remove(wc);
463     }
464 }
465
466 /*
-----
467  * Adds a search result to the searchResultsPanel
468  *
-----
469  */
470 private void showSearchResult(WordCell wc) {
471     synchronized(wcg) {
472         if (!wcg.contains(wc)) {
473             wcg.add(wc);
474             searchResultsPanel.add(wc);
475         }
476     }
477 }
478
479 /*
-----
480  * ComponentListener for setting all components to the right size after resizing
481  * the JFrame
482  *
-----
483  */
484 ComponentAdapter windowResizeListener = new ComponentAdapter() {
485     public void componentResized(ComponentEvent e) {
486         updateGUI(true);
487     }
488 };
489
490 /*
-----
491  * Updates the GUI. Resizes/moves components at the right sizes/positions.
492  *
-----
493  */
494 public void updateGUI(boolean full) {
495     Insets in = mainFrame.getInsets();
496     int width = mainFrame.getWidth() - in.left - in.right - 2*spacing;
497
498     textLayoutPanel.setSize(width, TEXT_LAYOUT_HEIGHT);
499     textLayoutPanel.setLocation(spacing, spacing);
500     wordPanel.setLocation(spacing, TEXT_LAYOUT_HEIGHT + 2*spacing);
501
502     int remainingHeight = mainFrame.getHeight() - in.top - in.bottom -
503     TEXT_LAYOUT_HEIGHT - 4*spacing;
504
505     wordPanel.setSize(width, remainingHeight/2);
506     searchPanel.setSize(width, remainingHeight/2);
507
508     searchPanel.setLocation(spacing, TEXT_LAYOUT_HEIGHT + wordPanel.getHeight()
509     + 3*spacing);

```

```

508
509 // Word panel elements
510 wordScrollPane.setLocation(spacing,
511                             (wordPanel.getHeight() - word.getHeight()) / 2);
512 defScrollPane.setSize(wordPanel.getWidth() - wordScrollPane.getWidth() -
513                       3*spacing,
514                       wordPanel.getHeight() - BUTTON_HEIGHT - TEXT_HEIGHT -
515                       3*spacing);
516 defScrollPane.setLocation(wordScrollPane.getWidth() + 2*spacing, TEXT_HEIGHT
517 + spacing);
518
519 int defPWidth = defScrollPane.getWidth()
520     - (defScrollPane.getVerticalScrollBar().isVisible()
521       ? defScrollPane.getVerticalScrollBar().getWidth()
522       : 0);
523 defPanel.setPreferredSize(new Dimension(defPWidth, allDefs.size() *
524 DEF_HEIGHT + BUTTON_HEIGHT + spacing));
525 inputWordLabel.setLocation(spacing, wordScrollPane.getY() - TEXT_HEIGHT -
526 TEXT_SPACING);
527 defLabel.setLocation(defScrollPane.getX(), TEXT_SPACING);
528
529 // Update the def locations
530 if (allDefs.size() != 0) {
531     for (int i = (full ? 0 : allDefs.size() - 1); i < allDefs.size(); i++) {
532         Def def = allDefs.get(i);
533         def.setSize(defPWidth, DEF_HEIGHT);
534         def.setLocation(0, i * def.getHeight());
535     }
536 }
537
538 addDef.setSize(defPWidth - 2*spacing, BUTTON_HEIGHT);
539 addDef.setLocation(spacing, allDefs.size() * DEF_HEIGHT + spacing);
540
541 // Buttons
542 int buttonWidth = (wordPanel.getWidth() - 6*spacing) / 5;
543 editEntry.setSize(buttonWidth, BUTTON_HEIGHT);
544 deleteEntry.setSize(buttonWidth, BUTTON_HEIGHT);
545 saveEntry.setSize(buttonWidth, BUTTON_HEIGHT);
546 deleteSelectedDefinitions.setSize(buttonWidth, BUTTON_HEIGHT);
547 saveChanges.setSize(buttonWidth, BUTTON_HEIGHT);
548
549 int baseHeight = wordPanel.getHeight() - BUTTON_HEIGHT - spacing;
550 editEntry.setLocation(spacing, baseHeight);
551 saveEntry.setLocation(buttonWidth + 2*spacing, baseHeight);
552 saveChanges.setLocation(2*buttonWidth + 3*spacing, baseHeight);
553 deleteSelectedDefinitions.setLocation(3*buttonWidth + 4*spacing, baseHeight);
554 deleteEntry.setLocation(4*buttonWidth + 5*spacing, baseHeight);
555
556 // Search panel elements
557 searchButton.setLocation(spacing, TEXT_HEIGHT + spacing);
558 searchScrollPane.setSize(searchPanel.getWidth() - searchButton.getWidth()
559 - 3*spacing, searchButton.getHeight());
560 searchScrollPane.setLocation(searchButton.getWidth() + 2*spacing,
561 TEXT_HEIGHT + spacing);
562
563 searchResultsScrollPane.setSize(searchPanel.getWidth() - 2*spacing,
564 searchPanel.getHeight() -
565 searchButton.getHeight() - TEXT_HEIGHT -
566 3*spacing);
567 searchResultsScrollPane.setLocation(spacing, TEXT_HEIGHT +
568 searchButton.getHeight() + 2*spacing);
569
570 // Update the search list locations
571 synchronized(wcg) {
572     ArrayList<WordCell> wcList = wcg.getGroupArrayList();
573
574     int searchResultWidth = searchResultsScrollPane.getWidth()
575     - (defScrollPane.getVerticalScrollBar().isVisible()
576       ? searchResultsScrollPane.getVerticalScrollBar().getWidth()
577       : 0);
578     searchResultsPanel.setPreferredSize

```



```

572         (new Dimension(searchResultWidth, WORD_CELL_HEIGHT * ((wcg.size() +
573           1) / 2)));
574     if (wcList.size() != 0) {
575         int wordCellWidth = searchResultWidth / 2;
576         for (int i = (full ? 0 : wcList.size() - 1); i < wcList.size(); i++) {
577             wcList.get(i).setLocation((i/2 != i/2.0 ? wordCellWidth : 0),
578                 WORD_CELL_HEIGHT * (i / 2));
579             wcList.get(i).setSize(wordCellWidth, WORD_CELL_HEIGHT);
580         }
581     }
582     mainFrame.repaint();
583 }
584
585 /*
586 -----
587 * Action listeners
588 *
589 -----
590 */
591 * Sets the action for each button.
592 */
593 ActionListener buttonActionListener = new ActionListener() {
594     @Override
595     public void actionPerformed(ActionEvent e) {
596         new Thread("Button action Thread") {
597             @Override
598             public void run() {
599                 JButton source = (JButton) e.getSource();
600
601                 if (source == addDef) {
602                     addDef(new Def(spacing));
603
604                 } else if (source == bold) {
605                     insertText("<b>", "</b>", "insert bold text here");
606
607                 } else if (source == italic) {
608                     insertText("<i>", "</i>", "insert italic text here");
609
610                 } else if (source == underline) {
611                     insertText("<u>", "</u>", "insert underlined text here");
612
613                 } else if (source == link) {
614                     insertText("<a href=\"destination\">", "</a>", "insert
615                         description here");
616
617                 } else if (source == editEntry) {
618                     prevFocusedDefField = null;
619                     WordCell wc = wcg.getLastSeleted();
620
621                     if (wc != null && wc.isSelected()) {
622                         // TODO
623                         // prompt for saving iff another entry is here already
624
625                         clearDefs();
626                         String text = wc.getText().replaceAll(";", ":");
627
628                         synchronized(definitions) {
629                             WordDef wordDef = definitions.get(text);
630
631                             word.setText(text);
632
633                             for (int i = 0; i < WordDef.TOTAL_TYPES; i++) {
634                                 ArrayList<String> defs = wordDef.getDef(i);
635
636                                 if (i == WordDef.OTHER) {
637                                     String[] descrNames = Def.defDescr;

```

```

638
639         for (int j = 0; j < defs.size(); j++) {
640             boolean descrFound = false;
641             String def = defs.get(j);
642
643             for (int k = 0; k < descrNames.length;
644                 k++) {
645                 if (def.length() <
646                     descrNames[k].length() + 2) continue;
647
648                 if (def.substring(0,
649                     descrNames[k].length() +
650                     2).equals(descrNames[k] + ": ")) {
651                     descrFound = true;
652                     addDef(new Def(WordDef.OTHER, k,
653                         def.substring(descrNames[k].length()
654                             + 2), spacing));
655                 }
656             }
657             if (!descrFound) {
658                 addDef(new Def(WordDef.OTHER, 0,
659                     def, spacing));
660             }
661         }
662     } else {
663         ArrayList<String> values =
664             wordDef.getValues(i);
665
666         for (int j = 0; j < defs.size(); j++) {
667             addDef(new Def(i + WordDef.FIRST_TYPE,
668                 WordDef.DEF, defs.get(j), spacing));
669         }
670
671         for (int j = 0; j < values.size(); j++) {
672             addDef(new Def(i + WordDef.FIRST_TYPE,
673                 WordDef.VALUE, values.get(j), spacing));
674         }
675     }
676 }
677
678 updateGUI(true);
679
680 }
681
682 } else if (source == saveEntry) {
683     prevFocusedDefField = null;
684     saveCurrentEntity();
685
686 } else if (source == saveChanges) {
687     prevFocusedDefField = null;
688
689     synchronized(definitions) {
690         Set<Map.Entry<String, WordDef>> defSet =
691             definitions.entrySet();
692         Iterator<Map.Entry<String, WordDef>> defIterator =
693             defSet.iterator();
694
695         boolean succes = false;
696
697         synchronized(canClose) {
698             waitForWriteClose();
699             canClose = false;
700
701             while (!succes) {
702                 defFile.delete();
703
704                 try (BufferedWriter bw = new BufferedWriter(new
705                     FileWriter(defFile))) {
706                     while (defIterator.hasNext()) {
707                         Map.Entry<String, WordDef> entry =

```



```

698         (Map.Entry<String, WordDef>)
699         defIterator.next();
700
701         bw.write(entry.getKey() + ";");
702
703         WordDef we = entry.getValue();
704
705         for (int i = 0; i < WordDef.TOTAL_TYPES;
706             i++) {
707             ArrayList<String> defs = we.getDef(i);
708             ArrayList<String> values =
709                 we.getValues(i);
710
711             boolean firstType = i !=
712                 WordDef.OTHER;
713
714             for (int j = 0; j < defs.size();
715                 j++) {
716                 if (defs.get(j).equals(""))
717                     continue;
718
719                 if (firstType) {
720                     firstType = false;
721                     bw.write("**" +
722                         WordDef.getTypeString(i) +
723                         ";");
724                 }
725                 bw.write(defs.get(j) + ";");
726             }
727
728             boolean firstValue = true;
729
730             for (int j = 0; j < values.size();
731                 j++) {
732                 if (values.get(j).equals(""))
733                     continue;
734
735                 if (firstType) {
736                     firstType = false;
737                     bw.write("**" +
738                         WordDef.getTypeString(i) +
739                         ";");
740                 }
741
742                 if (firstValue) {
743                     firstValue = false;
744                     bw.write("**VALUE;");
745                 }
746
747                 bw.write(values.get(j) + ";");
748             }
749
750             bw.write("; " +
751                 System.getProperty("line.separator"));
752         }
753         succes = true;
754     } catch (IOException ex) {
755         Log.write(ex);
756
757         if (!(JOptionPane
758             .showConfirmDialog(mainFrame,
759                 "Cannot change
760                 file. Please,
761                 close any
762                 applications
763                 using the file.
764                 Do you want to
765                 continue?",
766                 "Write error",
767                 JOptionPane.YES_N

```

```

749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810

O_OPTION,

JOptionPane.QUEST
ION_MESSAGE) ==
JOptionPane.YES_O
PTION)) {

        succes = true;
    }
}

canClose = true;
}
}

} else if (source == deleteSelectedDefinitions) {
    prevFocusedDefField = null;

    if (JOptionPane
        .showConfirmDialog(mainFrame,
            "Do you really want to delete the
            selected definition(s)?",
            "Delete",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE) ==
            JOptionPane.YES_OPTION) {

        for (int i = allDefs.size() - 1; i >= 0; i--) {
            Def def = allDefs.get(i);

            if (def.isSelected()) {
                removeDef(def);
            }
        }
    }

} else if (source == deleteEntry) {
    prevFocusedDefField = null;

    if (JOptionPane
        .showConfirmDialog(mainFrame,
            "Do you really want to delete
            this entry?", "Delete",
            JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE) ==
            JOptionPane.YES_OPTION) {

        WordCell wc = wcg.getLastSeleted();

        if (wc != null && wc.isSelected()) {
            synchronized(definitions) {
                definitions.remove(wc.getText().replaceAll(";",
                    " :"));
                removeSearchResult(wc);
            }
        }

        updateGUI(true);
    }

} else if (source == searchButton) {
    search();
} else {
    System.err.println("Un-idenified button press: " + source);
    return;
}

updateGUI(true);
}
}.start();
}

```

```

811     };
812
813     /*
-----
814     * This method blocks till the writer is finished writing
815     *
-----
816     */
817     public void waitForWriteClose() {
818         while (true) {
819             synchronized(canClose) {
820                 if (canClose != null && canClose) break;
821             }
822
823             try {
824                 Thread.sleep(10);
825
826             } catch (InterruptedException e) {
827                 Log.write(e);
828                 e.printStackTrace();
829             }
830         }
831     }
832
833     /*
-----
834     * Inserts text at the last used input field.
835     * Can insert optional text after the starting selection and before the ending
836     * selection
837     * iff the start and the end of a selection are equal.
838     *
-----
839     */
840     private void insertText(String front, String back, String... optional) {
841         if (prevFocusedDefField != null) {
842             synchronized(prevFocusedDefField) {
843                 synchronized(startSelection) {
844                     synchronized(endSelection) {
845                         // Remove the caret listener to avoid incorrect intermediate
846                         // updates
847                         prevFocusedDefField.removeCaretListener(defCaretListener);
848
849                         boolean noSelection = startSelection.equals(endSelection);
850
851                         prevFocusedDefField.insert(back, endSelection);
852                         prevFocusedDefField.insert(front, startSelection);
853
854                         startSelection += front.length();
855                         endSelection += front.length();
856
857                         if (noSelection && optional != null) {
858                             for (int i = 0; i < optional.length; i++) {
859                                 prevFocusedDefField.insert(optional[i], endSelection);
860                                 endSelection += optional[i].length();
861                             }
862                         }
863                         prevFocusedDefField.requestFocus();
864                         prevFocusedDefField.setCaretPosition(startSelection);
865                         prevFocusedDefField.moveCaretPosition(endSelection);
866
867                         // Restore the caret listener
868                         prevFocusedDefField.addCaretListener(defCaretListener);
869                     }
870                 }
871             }
872         }
873     }

```

```

874  /*
875  -----
876  * Saves the current showing entity.
877  *
878  * Surpressed warnings note:
879  * - Deprecation:
880  * The methods getTypeSelector(), getDefValueSelector(), getDefDescrSelector()
881  * and getDescrField()
882  * of the class building.Def are deprecated, and are allowed (and meant to) be
883  * used in this class
884  * and function ONLY.
885  * - Unchecked:
886  * Used to create an array of ArrayLists
887  *
888  -----
889  */
890  @SuppressWarnings({"deprecation", "unchecked"})
891  public void saveCurrentEntity() {
892      String name = word.getText().replaceAll(";", ":");
893
894      if (name == null || name.equals("")) return;
895
896      synchronized(allWordCells) {
897          synchronized(definitions) {
898              // Checks if the name is already used
899              if (definitions.get(name) != null) {
900                  // TODO:
901                  // Notify user of removal this
902                  definitions.remove(name);
903
904              } else {
905                  addSearchResult(name);
906              }
907
908              // Create a new definition
909              ArrayList<String>[] newDefs = (ArrayList<String>[]) new
910              ArrayList<WordDef.TOTAL_TYPES>();
911              ArrayList<String>[] newValues = (ArrayList<String>[]) new
912              ArrayList<WordDef.TOTAL_TYPES>();
913
914              for (int i = 0; i < newDefs.length; i++) {
915                  newDefs[i] = new ArrayList<String>();
916                  newValues[i] = new ArrayList<String>();
917              }
918
919              // Fill in the definitions and values
920              for (int i = 0; i < allDefs.size(); i++) {
921                  Def def = allDefs.get(i);
922                  if (def == null) continue;
923
924                  int type = WordDef.getTypeFromString
925                  ((String) def.getTypeSelector().getSelectedItem());
926
927                  if (type == WordDef.OTHER) {
928                      String addDefDescr = (String)
929                      def.getDefDescrSelector().getSelectedItem();
930
931                      if (addDefDescr.toUpperCase().equals("NONE")) {
932                          addDefDescr = "";
933
934                      } else {
935                          addDefDescr += ": ";
936                      }
937
938                      newDefs[type].add(addDefDescr +
939                      def.getDescrField().getText().replaceAll(";", ":"));
940
941                  } else {
942                      int defValue = WordDef.getDefValueFromString
943                      ((String) def.getDefValueSelector().getSelectedItem());

```

```

937         System.out.println(defValue);
938         System.out.println((String)
          def.getDefValueSelector().getSelectedItem());
939
940         if (defValue == WordDef.DEF) {
941
942             newDefs[type].add(def.getDescrField().getText().replaceAll
              (";", ":"));
943
944             } else { // defVaue == WordDef.VALUE
945
946                 newValues[type].add(def.getDescrField().getText().replaceA
              ll("; ", ":"));
947
948             }
949
950         definitions.put(name, new WordDef(newDefs, newValues));
951         advisor.updateMaxWordLength( name.split(" ").length);
952
953         synchronized(wcg) {
954             wcg.sortWordCells();
955         }
956     }
957
958     updateGUI(true);
959 }
960
961 /*
-----
962  * When the definitions are reloaded, reload them in the editor.
963  *
-----
964 */
965 public void updateDefinitions(Hashtable<String, WordDef> defs) {
966     synchronized(definitions) {
967         synchronized(defs) {
968             definitions = defs;
969         }
970     }
971 }
972
973 /*
-----
974  * Checks which textfield was selected as last.
975  * Alos keeps track of the cursor position and selection.
976  *
-----
977 */
978 CaretListener defCaretListener = new CaretListener() {
979     @Override
980     public void caretUpdate(CaretEvent e) {
981         JTextArea field = (JTextArea) e.getSource();
982         prevFocusedDefField = field;
983         startSelection = new Integer(field.getSelectionStart());
984         endSelection = new Integer(field.getSelectionEnd());
985     }
986 };
987
988 /*
-----
989  * Notifies the Advisor class that the editor was closed and
990  * waits untill all writers are finished.
991  *
-----
992 */

```

```

993 WindowAdapter windowClosed = new WindowAdapter() {
994     @Override
995     public void windowClosing(WindowEvent windowEvent) {
996         advisor.notifyCloseDefEditor();
997         waitForWriteClose();
998     }
999 };
1000
1001 /*
-----
1002  * Sets the visibility of the mainFrame
1003  *
-----
1004  */
1005 public void setVisible(boolean visible) {
1006     synchronized(mainFrame) {
1007         mainFrame.setVisible(visible);
1008     }
1009 }
1010
1011 /*
-----
1012  * CaretListener for setting the caret to the beginning of the JTextArea textInput
1013  *   iff the initial text is showing
1014  *
-----
1015  */
1016 CaretListener textAreaCaretListener = new CaretListener() {
1017     @Override
1018     public void caretUpdate(CaretEvent e) {
1019         Runnable r = new Runnable() {
1020             @Override
1021             public void run() {
1022                 if (!textTyped) {
1023                     searchBar.removeCaretListener(textAreaCaretListener);
1024                     searchBar.setCaretPosition(0);
1025                     searchBar.addCaretListener(textAreaCaretListener);
1026                 }
1027             }
1028         };
1029         SwingUtilities.invokeLater(r);
1030     }
1031 };
1032
1033 /*
-----
1034  * DocumentListener for setting/removing the initial text in the JTextArea
1035  *   searchBar
1036  *
-----
1037  */
1038 DocumentListener textAreaDocListener = new DocumentListener() {
1039     @Override
1040     public void changedUpdate(DocumentEvent e) { }
1041
1042     @Override
1043     public void insertUpdate(DocumentEvent e) {
1044         Runnable r = new Runnable() {
1045             @Override
1046             public void run() {
1047                 synchronized(searchBar) {
1048                     String text = searchBar.getText();
1049
1050                     if (!textTyped) {
1051                         textTyped = true;

```

```

        searchBar.getDocument().removeDocumentListener(textAreaDoc
1053         Listener);
        searchBar.setText(text.substring(0, text.length() -
1054         searchBeginInputText.length()));
        searchBar.setForeground(Color.BLACK);

        searchBar.getDocument().addDocumentListener(textAreaDocLis
1055         tener);

1056
1057         textTyped = true;
1058     }
1059 }
1060 }
1061 }
1062 };
1063     SwingUtilities.invokeLater(r);
1064 }
1065
1066 @Override
1067 public void removeUpdate(DocumentEvent e) {
1068     Runnable r = new Runnable() {
1069         @Override
1070         public void run() {
1071             synchronized(searchBar) {
1072                 String text = searchBar.getText();
1073
1074                 if (text.equals("")) {
1075
1076                     searchBar.getDocument().removeDocumentListener(textAreaDoc
1077                     Listener);
1078                     searchBar.setText(searchBeginInputText);
1079                     searchBar.setForeground(Color.GRAY);
1080
1081                     searchBar.getDocument().addDocumentListener(textAreaDocLis
1082                     tener);
1083
1084                     textTyped = false;
1085                 }
1086             }
1087         }
1088     };
1089     SwingUtilities.invokeLater(r);
1090 }
1091 };
1092
1093 /*
1094  * Select the wordCells that match the search text.
1095  */
1096 public void search() {
1097     prevFocusedDefField = null;
1098
1099     String searchText = searchBar.getText();
1100
1101     synchronized(allWordCells) {
1102         synchronized(wcg) {
1103             ArrayList<WordCell> wcs = wcg.getGroupArrayList();
1104
1105             for (int i = wcs.size() - 1; i >= 0; i--) {
1106                 hideSearchResult(wcs.get(i));
1107             }
1108
1109             for (int i = 0; i < allWordCells.size(); i++) {
1110                 WordCell wc = allWordCells.get(i);
1111
1112                 if (searchText.equals(searchBeginInputText)) {
1113                     showSearchResult(wc);
1114                 } else {
1115                     String text = wc.getText();
1116
1117                     if (text.length() >= searchText.length()) {
1118                         if (text.substring(0, searchText.length()).toUpperCase()

```



```

1117         .equals(searchText.toUpperCase())) {
1118             showSearchResult(wc);
1119         }
1120     }
1121 }
1122 }
1123 }
1124 }
1125 }
1126     updateGUI(true);
1127 }
1128 }
1129 }
1130     updateGUI(true);
1131 }
1132 }
1133 /*
1134  * Null action for disabling buttons via key maps.
1135  */
1136 Action nullAction = new AbstractAction() {
1137     @Override
1138     public void actionPerformed(ActionEvent e) { }
1139 };
1140 }
1141 /*
1142  * TMP
1143  */
1144 public static void setup(Advisor advisor, Hashtable<String, WordDef> def) {
1145     new DefFileEditor(advisor,
1146         new File(System.getProperty("user.dir") +
1147             "\\building\\definitions.csv"),
1148         new File(System.getProperty("user.dir") +
1149             "\\building\\definitions.csv.bac"), def, 100, 100);
1150 }
1151 }
1152 }
1153 /*
1154  * To initiate the program
1155  */
1156 }
1157 }
1158 }

```


11.8 Appendix G: building.editor.Def

```
1
2  /* * * * * * * * * * *
3   * Created by MASdude   *
4   *   Last modified:    *
5   *   19-10-2017       *
6   *   (dd-mm-yyyy)     *
7   * * * * * * * * * */
8
9  package building.editor;
10
11  // Own packages
12  import building.Advisor;
13  import building.WordDef;
14
15  // Java packages
16  import java.awt.Color;
17  import java.awt.Container;
18  import java.awt.Dimension;
19  import java.awt.Graphics;
20  import java.awt.Graphics2D;
21  import java.awt.event.ItemEvent;
22  import java.awt.event.ItemListener;
23
24  import java.io.File;
25  import java.io.IOException;
26
27  import javax.swing.BorderFactory;
28  import javax.swing.ComboBoxModel;
29  import javax.swing.JCheckBox;
30  import javax.swing.JComboBox;
31  import javax.swing.JPanel;
32  import javax.swing.JScrollPane;
33  import javax.swing.JTextArea;
34
35
36  public class Def extends JPanel {
37      final public static int TOTAL_HEIGHT = 30;
38      final public static int COMP_HEIGHT = 25;
39      final public static int SPACING_TOP = 2;
40      final public static int SPACING_LEFT = 3;
41
42      final public static String[] defDescr = {
43          "NONE", "linguistic", "legal",
44          "geometrical", "structural", "building physical",
45          "material technical", "financial", "aesthetics"
46      };
47
48
49      private static String[] types = null;
50      private static String[] defVal = null;
51
52      private JComboBox<String> typeSelector;
53      private JComboBox<String> defValueSelector;
54      private JComboBox<String> defDescrSelector;
55      private JScrollPane descrFieldScrollPane;
56      private JTextArea descrField;
57      private JCheckBox checkBox;
58
59      private int spacing = 0;
60
61      public Def(int spacing) {
62          this(0, spacing);
63      }
64
65      public Def(int width, int spacing) {
66          this(WordDef.OTHER, WordDef.DEF, "", width, spacing);
67      }
68
69      public Def(int type, int defValue, String text, int spacing) {
70          this(type, defValue, text, 0, spacing);
71      }
72
73      public Def(int type, int devValue, String text, int width, int spacing) {
```

```

74     super(null);
75     this.spacing = spacing;
76
77     if (types == null) {
78         types = new String[WordDef.TOTAL_TYPES];
79         for (int i = 0; i < types.length; i++) {
80             types[i] = WordDef.getTypeString(WordDef.FIRST_TYPE + i);
81         }
82     }
83
84     if (defVal == null) {
85         defVal = new String[] {
86             WordDef.getDevValueString(WordDef.DEF),
87             WordDef.getDevValueString(WordDef.VALUE)
88         };
89     }
90
91
92     typeSelector = new JComboBox<String>(types);
93     defValueSelector = new JComboBox<String>(defVal);
94     defDescrSelector = new JComboBox<String>(defDescr);
95     descrField = new JTextArea(text);
96     checkBox = new JCheckBox();
97
98     descrFieldScrollPane = new JScrollPane(descrField);
99
100    descrFieldScrollPane.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLL
    LBAR_NEVER);
101
102    descrFieldScrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR
    _NEVER);
103
104    typeSelector.setSelectedIndex(type);
105    if (type == WordDef.OTHER) {
106        defDescrSelector.setSelectedIndex(devValue - WordDef.FIRST_TYPE);
107    } else {
108        defValueSelector.setSelectedIndex(devValue - WordDef.FIRST_TYPE);
109    }
110
111    this.add(typeSelector);
112    this.add((type == WordDef.OTHER
113        ? defDescrSelector
114        : defValueSelector));
115    this.add(descrFieldScrollPane);
116    this.add(checkBox);
117
118
119    //typeSelector.setBackground(new Color(255, 255, 255));
120    //defValueSelector.setBackground(new Color(255, 255, 255));
121    //defDescrSelector.setBackground(new Color(255, 255, 255));
122    checkBox.setOpaque(false);
123
124    //this.setBackground(new Color(240, 240, 240));
125    this.setBounds(0, 0, width, 0);
126
127    descrFieldScrollPane.setBorder(BorderFactory.createLoweredBevelBorder());
128
129    // Add the item listeners
130    typeSelector.addItemListener(itemListener);
131 }
132
133 /*
-----
134  * Get functions
135  *
-----
136 */
137 /*
138  * Returns the type selector

```

```

139     * !!! ONLY USE THIS WHEN YOU KNOW WHAT YOU ARE DOING !!!
140     * Do NOT modify the selector!
141     * READ-ONLY!
142     */
143     @Deprecated
144     public JComboBox<String> getTypeSelector() {
145         return typeSelector;
146     }
147
148     /*
149     * Returns the def/value selector
150     * !!! ONLY USE THIS WHEN YOU KNOW WHAT YOU ARE DOING !!!
151     * Do NOT modify the selector!
152     * READ-ONLY!
153     */
154     @Deprecated
155     public JComboBox<String> getDefValueSelector() {
156         return defValueSelector;
157     }
158
159     /*
160     * Returns the def description selector
161     * !!! ONLY USE THIS WHEN YOU KNOW WHAT YOU ARE DOING !!!
162     * Do NOT modify the selector!
163     * READ-ONLY!
164     */
165     @Deprecated
166     public JComboBox<String> getDefDescrSelector() {
167         return defDescrSelector;
168     }
169
170     /*
171     * Returns the description field
172     */
173     public JTextArea getDescrField() {
174         return descrField;
175     }
176
177     /*
178     * Returns true if the checkbox is checked.
179     * False otherwise.
180     */
181     public boolean isSelected() {
182         return checkBox.isSelected();
183     }
184
185     /*
186     -----
187     * Set functions
188     *
189     -----
190
191     */
192     /*
193     * Sets the selected type.
194     */
195     public void setType(int i) {
196         typeSelector.getModel().setSelectedItem(i - WordDef.FIRST_TYPE);
197     }
198
199     /*
200     * Sets theselected def/value.
201     */
202     public void setDefValue(int i) {
203         defValueSelector.getModel().setSelectedItem(i - WordDef.DEF);
204     }
205
206     /*
207     * Sets the spacing
208     */
209     public void setSpacing(int spacing) {
210         this.spacing = spacing;

```

```

208         this.setBounds(this.getX(), this.getY(), this.getWidth(), this.getHeight());
209     }
210
211     /*
-----
212     * Update and action functions
213     *
-----
214     */
215     @Override
216     public void setBounds(int x, int y, int width, int height) {
217         super.setBounds(x, y, width, TOTAL_HEIGHT);
218
219         typeSelector.setSize(250, COMP_HEIGHT);
220         defValueSelector.setSize(150, COMP_HEIGHT);
221         defDescrSelector.setSize(150, COMP_HEIGHT);
222         checkBox.setSize(COMP_HEIGHT, COMP_HEIGHT);
223         descrFieldScrollPane.setSize(width - typeSelector.getWidth() -
224                                     defValueSelector.getWidth()
225                                     - checkBox.getWidth() - 3*spacing -
226                                     SPACING_LEFT, 20);
227
228         typeSelector.setLocation(SPACING_LEFT, SPACING_TOP);
229         defValueSelector.setLocation(typeSelector.getWidth() + spacing, SPACING_TOP);
230         defDescrSelector.setLocation(typeSelector.getWidth() + spacing, SPACING_TOP);
231         descrFieldScrollPane.setLocation(typeSelector.getWidth() +
232                                         defValueSelector.getWidth() + 2*spacing, 4);
233         checkBox.setLocation(typeSelector.getWidth() + defDescrSelector.getWidth()
234                             + descrFieldScrollPane.getWidth() + 3*spacing, 0);
235     }
236
237     @Override
238     protected void paintBorder(Graphics g) {
239         Graphics2D g2d = (Graphics2D) g;
240
241         // Left
242         g2d.drawLine(0, 0, 0, getHeight());
243         // Right
244         g2d.drawLine(getWidth() - 1, 0, getWidth() - 1, getHeight() - 1);
245         // Down
246         g2d.drawLine(0, getHeight() - 1, getWidth() - 1, getHeight() - 1);
247     }
248
249     /*
250     * Checks for changes in the selection of the first type selector.
251     * Changes the second selector accordingly.
252     */
253     ItemListener itemListener = new ItemListener() {
254         @Override
255         public void itemStateChanged(ItemEvent e) {
256             if (e.getStateChange() == ItemEvent.SELECTED) {
257                 if (e.getItem().equals(types[0])) {
258                     remove(defValueSelector);
259                     add(defDescrSelector);
260
261                 } else {
262                     add(defValueSelector);
263                     remove(defDescrSelector);
264                 }
265
266                 revalidate();
267                 repaint();
268             }
269         }
270     };
271
272     /*
-----
273     * To initiate the program

```

```
272      *  
      -----  
      -----  
273      */  
274      public static void main(String[] args) {  
275          new Advisor();  
276      }  
277  }  
278  
279
```

11.9 Appendix H: building.editor.Sort

```
1
2  /* * * * * * * * * * *
3   * Created by MASdude   *
4   *   Last modified:    *
5   *   19-10-2017       *
6   *   (dd-mm-yyyy)     *
7   * * * * * * * * * */
8
9  package building.editor;
10
11
12  public class Sort {
13      final public static int TYPE_LETTERS_Aa_Zz = 0;
14      final public static int TYPE_LETTERS_Zz_Aa = 1;
15      final public static int TYPE_NUM_9_0 = 2;
16      final public static int TYPE_NUM_0_9 = 3;
17
18      final public static int HINT_OTHER_NUM_LETTERS = 4;
19      final public static int HINT_OTHER_LETTERS_NUM = 5;
20      final public static int HINT_NUM_OTHER_LETTERS = 6;
21      final public static int HINT_LETTERS_OTHER_NUM = 7;
22      final public static int HINT_NUM_LETTERS_OTHER = 8;
23      final public static int HINT_LETTERS_NUM_OTHER = 9;
24
25      final public static int HINT_LONG_FIRST = 10;
26      final public static int HINT_SHORT_FIRST = 11;
27
28      final private static int TYPE_OTHER = 0;
29      final private static int TYPE_NUM = 1;
30      final private static int TYPE_LETTER_CAPS = 2;
31      final private static int TYPE_LETTER_NO_CAPS = 3;
32
33
34      /*
35       * This is supposed to be a static function only class,
36       * so it only uses memory and overhead if you create an instance.
37       */
38      @Deprecated
39      public Sort() { }
40
41      /*
42       * Uses bubble sort to sort the array.
43       * The algorithm is in place, so the input array is modified.
44       * key determines the relative position of each element
45       * If an array of (not null) keys is given, return the sorted keys.
46       * Else return the array of sorted Strings.
47       */
48      public static String[] textBubbleSort(String[] inArray,
49                                          int typeLetters, int typeNum, int
                                          sortingHint, int lengthHint) {
50          return (String[]) textBubbleSort(inArray, null,
51                                          typeLetters, typeNum, sortingHint,
52                                          lengthHint);
53      }
54
55      public static Object[] textBubbleSort(String[] inArray, Object[] meta,
56                                          int typeLetters, int typeNum, int sortingHint,
57                                          int lengthHint) {
58          // Type and hint checking
59          if (typeLetters < 0 || typeLetters > 1)
60              throw new IllegalArgumentException("Invalld type for letters: " +
61              typeLetters);
62          if (typeNum < 2 || typeNum > 3)
63              throw new IllegalArgumentException("Invalld type for numbers: " +
64              typeNum);
65          if (sortingHint < 4 || sortingHint > 9)
66              throw new IllegalArgumentException("Invalld sorting hint: " +
67              sortingHint);
68          if (lengthHint < 10 || lengthHint > 11)
69              throw new IllegalArgumentException("Invalld length hint: " + lengthHint);
70
71          // Array checking
72          if (inArray == null)
```

```

68         throw new NullPointerException("\'inArray\' was null!");
69     if (meta != null && inArray.length != meta.length)
70         throw new IllegalArgumentException("Invalid array/position size: "
71                                         + inArray.length + " (array), " +
72                                         meta.length + " (pos).");
73
74     synchronized(inArray) {
75         String switchElem = null;
76         Object switchKey = null;
77         int start = 0;
78         int end = inArray.length;
79
80         // true -> 0 to end,
81         // false -> end to 0
82         boolean direction = true;
83         boolean changed = true;
84
85         while (changed) {
86             changed = false;
87
88             //System.out.println(start + ", " + end);
89             for (int i = (direction ? start : end - 2)
90                 ; (direction ? i < end - 1 : i >= start)
91                 ; i = (direction ? i+1 : i-1))
92             {
93                 if (!inOrder(inArray[i], inArray[i + 1], typeLetters, typeNum,
94                             sortingHint, lengthHint)) {
95                     switchElem = inArray[i];
96                     inArray[i] = inArray[i+1];
97                     inArray[i+1] = switchElem;
98
99                     if (meta != null) {
100                         switchKey = meta[i];
101                         meta[i] = meta[i + 1];
102                         meta[i + 1] = switchKey;
103                     }
104
105                     changed = true;
106                 }
107
108                 if (direction) {
109                     end--;
110                 } else {
111                     start++;
112                 }
113
114                 direction = !direction;
115             }
116         }
117
118         if (meta != null) {
119             return meta;
120         } else {
121             return (String[]) inArray;
122         }
123     }
124 }
125
126 /*
127  * Compares the two Strings using the sorting hint.
128  * Returns true iff str1 <= str2.
129  * False otherwise.
130  */
131 public static boolean inOrder(String str1, String str2,
132                             int typeLetters, int typeNum, int sortingHint, int
133                             lengthHint) {
134     // Check for equality
135     if (str1.equals(str2)) {
136         return true;
137     }

```

```

138     // Calculate the smallest length of the two Strings
139     int minSize = str1.length();
140     if (minSize > str2.length()) {
141         minSize = str2.length();
142     }
143
144     // Determine the order
145     int[] key = new int[2];
146     for (int i = 0; i < minSize; i++) {
147         key[0] = str1.charAt(i);
148         key[1] = str2.charAt(i);
149         if (key[0] == key[1]) continue;
150
151         // Determine the type
152         int[] type = {TYPE_OTHER, TYPE_OTHER};
153         for (int j = 0; j < 2; j++) {
154             if (key[j] >= '0' && key[j] <= '9') {
155                 type[j] = TYPE_NUM;
156
157             } else if (key[j] >= 'a' && key[j] <= 'z') {
158                 type[j] = TYPE_LETTER_NO_CAPS;
159
160             } else if (key[j] >= 'A' && key[j] <= 'Z') {
161                 type[j] = TYPE_LETTER_CAPS;
162             }
163         }
164
165         if (type[0] == type[1]) {
166             if (type[0] == TYPE_OTHER) {
167                 return (key[0] <= key[1]);
168
169             } else if (type[0] == TYPE_NUM) {
170                 if (typeNum == TYPE_NUM_0_9) {
171                     return (key[0] <= key[1]);
172
173                 } else { // typeNum == TYPE_NUM_9_0
174                     return (key[0] >= key[1]);
175                 }
176
177             } else if (type[0] == TYPE_LETTER_CAPS ||
178                 type[0] == TYPE_LETTER_NO_CAPS) {
179                 if (typeLetters == TYPE_LETTERS_Aa_Zz) {
180                     return (key[0] <= key[1]);
181
182                 } else { // typeLetters == TYPE_LETTERS_Zz_Aa
183                     return (key[0] >= key[1]);
184                 }
185             }
186
187         } else {
188             if (type[0] == TYPE_NUM) {
189                 if (sortingHint == HINT_NUM_OTHER_LETTERS ||
190                     sortingHint == HINT_NUM_LETTERS_OTHER) {
191                     return true;
192
193                 } else if (sortingHint == HINT_OTHER_LETTERS_NUM ||
194                     sortingHint == HINT_LETTERS_OTHER_NUM) {
195                     return false;
196
197                 } else if (sortingHint == HINT_LETTERS_NUM_OTHER) {
198                     return (type[1] == TYPE_OTHER);
199
200                 } else { // sortingHint == HINT_OTHER_NUM_LETTERS
201                     return (type[1] != TYPE_OTHER);
202                 }
203
204             } else if (type[0] == TYPE_LETTER_CAPS ||
205                 type[0] == TYPE_LETTER_NO_CAPS) {
206                 // Note that type[0] != type[1]
207                 if (type[1] == TYPE_LETTER_NO_CAPS) { // type[0] ==
208                     TYPE_LETTER_CAPS
209                     if (typeLetters == TYPE_LETTERS_Aa_Zz) {

```



```

210         return (key[0] - 'A' <= key[1] - 'a');
211
212     } else { // typeLetters == TYPE_LETTERS_Zz_Aa
213         return (key[0] - 'A' >= key[1] - 'a');
214     }
215
216     } else if (type[1] == TYPE_LETTER_CAPS) { // type[0] ==
TYPE_LETTER_NO_CAPS
217         if (typeLetters == TYPE_LETTERS_Aa_Zz) {
218             return (key[0] - 'a' < key[1] - 'A');
219
220         } else { // typeLetters == TYPE_LETTERS_Zz_Aa
221             return (key[0] - 'a' > key[1] - 'A');
222         }
223
224     } else {
225         if (sortingHint == HINT_LETTERS_OTHER_NUM ||
226             sortingHint == HINT_LETTERS_NUM_OTHER) {
227             return true;
228
229         } else if (sortingHint == HINT_OTHER_NUM_LETTERS ||
230             sortingHint == HINT_NUM_OTHER_LETTERS) {
231             return false;
232
233         } else if (sortingHint == HINT_OTHER_LETTERS_NUM) {
234             return (type[1] == TYPE_NUM);
235
236         } else { // sortingHint == HINT_NUM_LETTERS_OTHER
237             return (type[1] == TYPE_OTHER);
238         }
239     }
240
241     } else { // type[0] == TYPE_OTHER
242         if (sortingHint == HINT_OTHER_NUM_LETTERS ||
243             sortingHint == HINT_OTHER_LETTERS_NUM) {
244             return true;
245
246         } else if (sortingHint == HINT_NUM_LETTERS_OTHER ||
247             sortingHint == HINT_LETTERS_NUM_OTHER) {
248             return false;
249
250         } else if (sortingHint == HINT_NUM_OTHER_LETTERS) {
251             return (type[1] != TYPE_NUM);
252
253         } else { // sortingHint == HINT_LETTERS_OTHER_NUM
254             return (type[1] == TYPE_NUM);
255         }
256     }
257 }
258
259 // One String is longer and that whole string equals the first part of the
260 // other
261 // String. Now use the lengthHint to determine the result.
262
263 if (lengthHint == HINT_LONG_FIRST) {
264     return (str1.length() >= str2.length());
265
266 } else { // lengthHint == HINT_SHORT_FIRST
267     return (str1.length() <= str2.length());
268 }
269
270
271 /*
272 public static void main(String[] args) {
273     //String[] array = new String[] {"a", "b", "A", "B", "A", "1", "2", "#",
"Ab", "AB", "Aa"};
274     //String[] array = new String[] {"aa", "a", "AA", "A", "bb", "b", "BB", "B",
"15", "14", "13", "13", "12", "11", "2", "##", "$$"};
275     //String[] array = new String[] {"1", "#"};
276     /*
277     String[] array = new String[10000];
278     int min = 48;

```

```

279     int max = 122 - min + 1;
280     for (int i = 0; i < array.length; i++) {
281         array[i] = "" + ((char) (min + Math.floorMod(i, max)))
282             + ((char) (min + Math.floorMod(i / max, max)));
283     }*/
284
285     String[] array = {"b", "a", "12", "11", "$"};
286     String[] sorted = null;
287     long time1 = System.currentTimeMillis();
288     for (int i = 0; i < 1; i++) {
289         sorted = textBubbleSort(array, TYPE_LETTERS_Aa_Zz, TYPE_NUM_0_9,
290             HINT_OTHER_NUM_LETTERS, HINT_SHORT_FIRST);
291     }
292     long time2 = System.currentTimeMillis();
293     System.out.println("Time taken: " + (time2 - time1));
294
295     for (int i = 0; i < array.length; i++) {
296         System.out.println(array[i]);
297     }
298 }*/
299 }

```

11.10 Appendix I: building.editor.WordCell

```
1
2  /* * * * * * * * * * *
3   * Created by MASdude   *
4   *   Last modified:    *
5   *   19-10-2017        *
6   *   (dd-mm-yyyy)      *
7   * * * * * * * * * */
8
9  package building.editor;
10
11  // Own packages
12  import building.Advisor;
13
14  // Java packages
15  import java.awt.BasicStroke;
16  import java.awt.Color;
17  import java.awt.Graphics;
18  import java.awt.Graphics2D;
19
20  import java.io.File;
21  import java.io.IOException;
22
23  import javax.swing.BorderFactory;
24  import javax.swing.JPanel;
25  import javax.swing.border.Border;
26
27
28  public class WordCell extends JPanel {
29      final private static int BORDER_SPACING = 2;
30      final private static int FOCUS_BORDER_SPACING = 6;
31
32      private boolean hasFocus = false;
33      private boolean isSelected = false;
34      private String text;
35
36
37      /*
38      -----
39      * Constructor
40      *
41      -----
42      */
43      WordCell(String text) {
44          super();
45          this.text = text;
46          this.setFocus(false);
47          this.setSelected(false);
48      }
49
50      /*
51      -----
52      * Set Methods
53      *
54      -----
55      */
56
57      /*
58      * Selects this cell.
59      * This way of selecting a cell is deprecated when
60      * using multiple instances of WordCell. Use a WordCellGroup instead.
61      */
62      @Deprecated
63      public void setSelected(boolean select) {
64          isSelected = select;
65          this.repaint();
66      }
67
68      /*
69      * Sets the visible focus.
70      */
71  }
```

```

66     * Does NOT set the actual focus!
67     *
68     * This way of setting the visual focus is deprecated when
69     * using multiple instances of WordCell. Use a WordCellGroup instead.
70     */
71     @Deprecated
72     public void setFocus(boolean focus) {
73         hasFocus = focus;
74         this.repaint();
75     }
76
77
78     /*
79     -----
80     * Get methods
81     *
82     -----
83     */
84     /*
85     * Returns true if selected.
86     * False otherwise.
87     */
88     public boolean isSelected() {
89         return isSelected;
90     }
91
92     /*
93     * Returns true if this component shows focus.
94     * False otherwise.
95     */
96     public boolean hasFocus() {
97         return hasFocus;
98     }
99
100    /*
101    * Returns the text of the word
102    */
103    public String getText() {
104        return text;
105    }
106
107    /*
108    -----
109    * Paint methods
110    *
111    -----
112    */
113
114    @Override
115    protected void paintComponent(Graphics g) {
116        super.paintComponent(g);
117        Graphics2D g2d = (Graphics2D) g;
118
119        // Paint the background
120        g2d.setPaint(isSelected
121            ? new Color(0, 200, 255)
122            : new Color(255, 255, 255));
123        g2d.fillRect(0, 0, getWidth(), getHeight());
124
125        // Paint the text
126        int baseline = g2d.getFontMetrics().getAscent();
127        g2d.setPaint(new Color(0, 0, 0));
128        g2d.drawString(text, FOCUS_BORDER_SPACING + 3, FOCUS_BORDER_SPACING +
129            baseline);
130    }
131
132    @Override
133    protected void paintBorder(Graphics g) {
134        Graphics2D g2d = (Graphics2D) g;

```

```

130
131 // Paint the black solid border
132 g2d.setPaint(new Color(0, 0, 0));
133 g2d.drawRect(BORDER_SPACING, BORDER_SPACING,
134             getWidth()-1 - 2*BORDER_SPACING, getHeight()-1 -
135             2*BORDER_SPACING);
136
137 // Paint the dotted border iff visual focus
138 if (hasFocus) {
139     g2d.setStroke(new BasicStroke(1f, BasicStroke.CAP_BUTT,
140     BasicStroke.JOIN_BEVEL, 0, new float[]{1.5f}, 0));
141     g2d.drawRect(FOCUS_BORDER_SPACING, FOCUS_BORDER_SPACING,
142     getWidth()-1 - 2*FOCUS_BORDER_SPACING, getHeight()-1 -
143     2*FOCUS_BORDER_SPACING);
144 }
145
146 /*
147 -----
148 * To initiate the program
149 *
150 -----
151 */
152 public static void main(String[] args) {
153     //DefFileEditor.setup();
154     new Advisor();
155 }

```

11.11 Appendix J: building.editor.WordCellGroup

```
1
2  /* * * * * * * * * * *
3   * Created by MASdude   *
4   *   Last modified:    *
5   *   19-10-2017        *
6   *   (dd-mm-yyyy)      *
7   * * * * * * * * * */
8
9  package building.editor;
10
11  // Own packages
12  import building.Advisor;
13  import building.log.Log;
14
15  // Java packages
16  import java.awt.event.MouseAdapter;
17  import java.awt.event.MouseEvent;
18
19  import java.io.IOException;
20
21  import java.text.RuleBasedCollator;
22  import java.text.ParseException;
23
24  import java.util.ArrayList;
25
26
27  public class WordCellGroup {
28      final public static int TYPE_NONE = 0;
29      final public static int TYPE_SELECT_ONE_ONLY = 1;
30
31      private int type = TYPE_NONE;
32      private ArrayList<WordCell> wordCells;
33      private WordCell lastSelected = null;
34
35      public WordCellGroup(WordCell... wcs) {
36          this(TYPE_NONE, wcs);
37      }
38
39      public WordCellGroup(int type, WordCell... wcs) {
40          this.type = type;
41          wordCells = new ArrayList<WordCell>();
42
43          if (wcs != null) {
44              synchronized(wordCells) {
45                  for (int i = 0; i < wcs.length; i++) {
46                      if (wcs[i] != null) {
47                          wordCells.add(wcs[i]);
48                      }
49                  }
50              }
51          }
52      }
53
54      /*
55       * Focusses on the cell which was just clicked.
56       * Inverts the selection of that cell.
57       * If the cell was not yet selected, also deselects any other selections.
58       *
59       * Surpressed warnings note:
60       * - Deprecation:
61       * The methods setSelected() and setFocus() are deprecated, and are allowed
62       * (and meant to) be used in this class and function ONLY.
63       */
64      @SuppressWarnings("deprecation")
65      MouseAdapter oneSelectMouseListener = new MouseAdapter() {
66          @Override
67          public void mousePressed(MouseEvent e) {
68              WordCell source = (e.getSource() instanceof WordCell ? (WordCell)
69                  e.getSource() : null);
70
71              synchronized (wordCells) {
72                  for (int i = 0; i < wordCells.size(); i++) {
73                      WordCell wc = wordCells.get(i);
```

```

73
74         wc.setSelected(wc.equals(source) && !wc.isSelected());
75         wc.setFocus(wc.equals(source));
76
77         if (source.equals(wc) && wc.isSelected()) {
78             lastSelected = wc;
79         }
80     }
81 }
82
83 }
84 };
85
86 /*
87  * Sets the type
88  */
89 public void setType(int type) {
90     synchronized(wordCells) {
91         if (this.type != type) {
92             this.type = type;
93
94             for (int i = 0; i < wordCells.size(); i++) {
95                 wordCells.get(i).removeMouseListener(oneSelectMouseListener);
96             }
97
98             if (type == TYPE_SELECT_ONE_ONLY) {
99                 for (int i = 0; i < wordCells.size(); i++) {
100                     addRightListener(wordCells.get(i));
101                 }
102             }
103         }
104     }
105 }
106
107 /*
108  * Adds the correct listener for a WordCell.
109  */
110 private void addRightListener(WordCell wc) {
111     if (type == TYPE_SELECT_ONE_ONLY) {
112         wc.addMouseListener(oneSelectMouseListener);
113     }
114 }
115 /*
116  * Adds a wordCell to the group
117  */
118 public void add(WordCell wc) {
119     synchronized(wordCells) {
120         if (wc != null) {
121             wordCells.add(wc);
122             addRightListener(wc);
123         }
124     }
125 }
126
127 /*
128  * Removes a wordCell from the group
129  */
130 public WordCell remove(WordCell wc) {
131     synchronized(wordCells) {
132         wordCells.remove(wc);
133         wc.removeMouseListener(oneSelectMouseListener);
134     }
135
136     return wc;
137 }
138
139 /*
140  * Removes all wordCells from the list
141  */
142 public void clear() {
143     synchronized(wordCells) {
144         for (int i = 0; i < wordCells.size(); i++) {
145             wordCells.get(i).removeMouseListener(oneSelectMouseListener);

```

```

146         }
147
148         wordCells.clear();
149     }
150 }
151
152 /*
153  * Returns the number of wordCells in the group
154  */
155 public int size() {
156     return wordCells.size();
157 }
158
159 /*
160  * Returns all the wordCells contained in this group
161  */
162 public ArrayList<WordCell> getGroupArrayList() {
163     synchronized(wordCells) {
164         ArrayList<WordCell> copyWCS = new ArrayList<WordCell>(wordCells.size());
165         copyWCS.addAll(wordCells);
166         return copyWCS;
167     }
168 }
169
170 /*
171  * Returns the last item that was selected via this group
172  */
173 public WordCell getLastSeleted() {
174     return lastSelected;
175 }
176
177 /*
178  * Sorts the wordCells alphabetically
179  */
180 public void sortWordCells() {
181     synchronized(wordCells) {
182         WordCell[] keys = new WordCell[wordCells.size()];
183         String[] array = new String[wordCells.size()];
184         for (int i = 0; i < array.length; i++) {
185             keys[i] = wordCells.get(i);
186             array[i] = wordCells.get(i).getText();
187         }
188
189         Sort.textBubbleSort(array, keys,
190                             Sort.TYPE_LETTERS_Aa_Zz,
191                             Sort.TYPE_NUM_0_9,
192                             Sort.HINT_OTHER_NUM_LETTERS,
193                             Sort.HINT_SHORT_FIRST);
194
195         wordCells.clear();
196         for (int i = 0; i < array.length; i++) {
197             wordCells.add(keys[i]);
198         }
199     }
200 }
201
202 /*
203  * Returns true iff the WordCell is in this group.
204  * False otherwise.
205  */
206 public boolean contains(WordCell wc) {
207     synchronized(wordCells) {
208         return wordCells.contains(wc);
209     }
210 }
211
212 /*
213  * Returns tue iff the given name is already contained in
214  * some WordCell. False otherwise.
215  */
216 public boolean containsName(String name) {
217     synchronized(wordCells) {
218         for (int i = 0; i < wordCells.size(); i++) {

```



```

219         if (wordCells.get(i).getText().equals(name)) {
220             return true;
221         }
222     }
223     return false;
224 }
225 }
226 }
227
228
229  /*
-----
230  * To initiate the program
231  *
-----
232  */
233 public static void main(String[] args) {
234     new Advisor();
235 }
236 }

```