A DEEP LEARNING BASED APPROACH TO
SKETCH RECOGNITION AND MODEL
TRANSFORMATION FOR REQUIREMENTS
ELICITATION AND MODELLING

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2021

By
Oluwatoni Segun Olatunji
Department of Computer Science
Contents

Contents ........................................................................................................................................... 2
List of Figures .................................................................................................................................. 6
List of Tables ..................................................................................................................................... 12
Abstract ........................................................................................................................................... 13
Declaration .......................................................................................................................................... 14
Copyright ........................................................................................................................................ 15
Acknowledgement ........................................................................................................................... 16

Chapter 1  Introduction ..................................................................................................................... 17
  1.1 Research Problem ....................................................................................................................... 17
  1.2 Research Aim and Objectives ...................................................................................................... 20
  1.3 Research Contributions ............................................................................................................... 20
  1.4 Research Methodology .............................................................................................................. 21
  1.5 Overview of Thesis .................................................................................................................... 23

Chapter 2  Related Work ................................................................................................................... 24
  2.1 Introduction ............................................................................................................................... 24
  2.2 Sketch Recognition Approaches ............................................................................................... 25
     2.2.1 Gesture-based Sketch Recognition Approaches ................................................................. 25
     2.2.2 Stroke-based Sketch Recognition Approaches ............................................................... 26
     2.2.3 Geometry-based Sketch Recognition Approaches .......................................................... 34
     2.2.4 Image-based Sketch Recognition Approaches ............................................................... 38
     2.2.5 Summary ........................................................................................................................... 39
  2.3 Sketch Transformation Approaches ............................................................................................ 41
  2.4 Sketching Software Tools .......................................................................................................... 43
     2.4.1 Early Requirements Engineering (RE) Tools ....................................................................... 44
     2.4.2 General Modelling Tools ...................................................................................................... 45
     2.4.3 User Interface Design Tools .................................................................................................. 46
     2.4.4 General Sketching Tools ...................................................................................................... 47
     2.4.5 A Set of Common Features Supported by Related Sketch Tools ........................................ 48
Chapter 3  Foundations for iSketch ................................................................. 52
  3.1 Overview of iSketch ............................................................................. 52
  3.2 Sketch Recognition and Deep Neural Networks ................................. 54
    3.2.1 CNN Architecture ....................................................................... 57
    3.2.2 CNN Optimizations .................................................................... 59
  3.3 Model Transformation ......................................................................... 60
  3.4 DSL and Metamodel Construction ....................................................... 62
    3.4.1 DSL Construction ....................................................................... 62
    3.4.2 Metamodel Construction ............................................................... 65
  3.5 Summary ............................................................................................. 66

Chapter 4  iSketch Design ........................................................................ 67
  4.1 Introduction ........................................................................................ 67
  4.2 Sketch-Based Recognition Method ...................................................... 67
  4.3 Sketch-Based Model Transformation Method ...................................... 71
    4.3.1 Sketch-Based Modelling Language ........................................... 71
    4.3.2 Model Transformation Process Flow and Rules .......................... 78
    4.3.3 Properties of iSketch Sketch-Based Model Transformation Method .. 92

Chapter 5  iSketch Implementation ............................................................. 93
  5.1 Introduction ........................................................................................ 93
  5.2 User Interface and Related Components .............................................. 93
    5.2.1 Sketch Canvas ............................................................................ 94
    5.2.2 Sketch Annotator ....................................................................... 95
    5.2.3 Sketch Repository ..................................................................... 95
    5.2.4 Sketch Recognition Engine ....................................................... 95
  5.3 UI Meta-model and Associated Components ..................................... 97
    5.3.1 Symbol Class ............................................................................ 98
    5.3.2 Link Class ................................................................................ 102
    5.3.3 UCD Model Class ..................................................................... 103
  5.4 Sketch Model Transformer and Related Components ........................ 105
    5.4.1 Sketch Model Transformer .......................................................... 105
    5.4.2 Sketch Model Parser ................................................................. 105
    5.4.3 Sketch Transformation Engine ................................................... 106
    5.4.4 Sketch Model Interpreter ........................................................... 106

2.4.6 Features to be supported for iSketch .............................................. 50
2.5 Limitations of Related Work ............................................................... 51
A.3. Benchmark Models used to Evaluate iSketch Sketch-Based Model

Transformation Method ........................................................................................................ 175
NextGen Point of Sale (POS) System ................................................................................. 175
Elevator – An Elevator System .......................................................................................... 178
ATM - A Banking System .................................................................................................. 181

Word Count: 41143
List of Figures

Figure 1.1: Research Methodology [14]....................................................... 21

Figure 3.1: iSketch Architectural Design. It shows its main components and their respective interactions, Control Flow (solid arrows), Data Flow (dotted arrows)................................................................. 53

Figure 3.2: Twin Peaks showing the Relationship between iSketch Functional Requirements and its Architectural Components ................. 54

Figure 3.3: Examples of sketches (TU-Berlin [73]) ......................... 55

Figure 3.4: An example showing the challenges of sketch recognition [76]........................................................................................................ 56

Figure 3.5: A flow chart example of prior methods for sketch recognition ........................................................................................................ 56

Figure 3.6: A CNN Architecture example for Sketch Classification ....... 57

Figure 3.7: Top-down approach for constructing DSL ......................... 63

Figure 3.8: Example-driven approach for DSL construction adapted from [53]........................................................................................................... 64

Figure 4.1: Framework of Proposed Sketch-Based Recognition Method............................................................................................................. 68

Figure 4.2: Abstract Syntax of iSketch Sketch-Based Modelling Language........................................................................................................ 72

Figure 4.3: Concrete Syntax of iSketch Sketch-Based Modelling Language........................................................................................................ 73

Figure 4.4: Specification of Sketch-Based Modelling Language Vocabulary in Alloy........................................................................................... 75

Figure 4.5: Specification of Rules to Validate the Vocabulary of Sketch-Based Modelling Language........................................................................ 77
Figure 4.6: Specification of Operations to Check the Consistency of the Vocabulary of Sketch-Based Modelling Language ........................................ 77

Figure 4.7: Result showing the Consistency of the Vocabulary of Sketch-Based Modelling Language......................................................... 77

Figure 4.8: A Generated Instance Model of Sketch-Based Modelling Language............................................................................................... 78

Figure 4.9: Process Flow for Sketch-Based Model Transformation........ 78

Figure 4.10: The Structure of Stanford Parser Output in XML Schema .... 79

Figure 4.11: Transformed Structure of Stanford Parser Output in JSON Schema .................................................................................................. 80

Figure 4.12: An Example Showing Identification of Compound Name that form a Single Agent................................................................. 85

Figure 4.13: Identification of Group of Words that form a Single Agent........................................................................................................... 86

Figure 4.14: Extraction Rule 1 ........................................................................ 86

Figure 4.15: Extraction Rule 2 ........................................................................ 87

Figure 4.16: Extraction Rule 3 ........................................................................ 87

Figure 4.17: Extraction Rule 4 ........................................................................ 87

Figure 4.18: Translation Rule 1 ........................................................................ 87

Figure 4.19: Translation Rule 2 ........................................................................ 88

Figure 4.20: Translation Rule 3 ........................................................................ 88

Figure 4.21: Translation Rule 4 ........................................................................ 89

Figure 4.22: Translation Rule 5 ........................................................................ 90

Figure 5.1: iSketch User Interface. It shows the tool’s Sketch Canvas (A), Annotator (B), Sketch Repository (C), Model Transformer (D) and Other House-Keeping Tools. .................................................... 94

Figure 5.2: iSketch CNN Recognition Model.................................................. 96

Figure 5.3: Prediction of an Unknown Sketch................................................ 97

Figure 5.4: UI Meta-model and Associated Components of iSketch ......... 98
Figure 5.5: Code Snippet of Get Symbol function ......................................... 99

Figure 5.6: Code Snippet of Get Symbol function ....................................... 100

Figure 5.7: An example showing the input and output of the Sketch Model Parser component ............................................................... 106

Figure 5.8: A section of iSketch. It shows a dragged and dropped use case symbol .................................................................................... 109

Figure 5.9: A section of iSketch. It shows the change of name from use case to “withdraw funds” ................................................................. 110

Figure 5.10: A section of iSketch. It shows a Sketch Canvas for creating custom symbols ................................................................. 110

Figure 5.11: A section of iSketch. It shows the custom symbol “customer” newly created by the user ...................................................... 111

Figure 5.12: A section of iSketch. It shows the actor “customer” added to the Sketch Canvas ................................................................. 111

Figure 5.13: A section of iSketch. It shows the created association relationship between the actor (customer) and use case (withdraw funds) ...................................................................................... 111

Figure 5.14: A section of iSketch. It shows added attributes of the actor (customer). .................................................................................. 112

Figure 5.15: A section of iSketch. It shows adding of natural language text details to the use case (withdraw funds) .............................................. 113

Figure 5.16: A section of iSketch. It shows the saved use case diagram (bank atm system) ........................................................................ 113

Figure 5.17: It shows a Sequence diagram software model generated from the Use Case diagram in Figure 5.16 .............................................. 114

Figure 5.18: It shows a Class diagram conceptual model generated from the Use Case diagram in Figure 5.16 .............................................. 115

Figure 6.1: Some examples of sketch images in iSketch dataset used to evaluate iSketch Sketch-Based Recognition method .................................. 117

Figure 6.2: iSketch Sketch Recognition Result on iSketch Dataset without Colour Inversion ........................................................................ 119
Figure 6.3: iSketch Sketch Recognition Result on iSketch Dataset without Colour Inversion ........................................................................................................ 120

Figure 6.4: Recall and Precision values of Source Agents Generated by iSketch ........................................................................................................... 124

Figure 6.5: Recall and Precision values of Target Agents Generated by iSketch ........................................................................................................ 125

Figure 6.6: Recall and Precision values of Message Interaction between Agents Generated by iSketch ........................................................................ 125

Figure 6.7: Recall and Precision values of Classes Generated by iSketch ........................................................................................................... 127

Figure 6.8: Recall and Precision values of Classes’ Methods Generated by iSketch ............................................................................................... 127

Figure 6.9: Recall and Precision values of Classes’ Association and Generation Relationships Generated by iSketch ........................................... 128

Figure 0.1: Sequence Diagram Generated by iSketch for Process Cash Sale (NextGen Point of Sale System) ................................................................. 156

Figure 0.2: Class Diagram Generated by iSketch for Process Cash Sale (NextGen Point of Sale System) ................................................................. 157

Figure 0.3: Sequence Diagram Generated by iSketch for Process Credit Sale (NextGen Point of Sale System) ................................................................. 157

Figure 0.4: Class Diagram Generated by iSketch for Process Credit Sale (NextGen Point of Sale System) ................................................................. 158

Figure 0.5: Sequence Diagram Generated by iSketch for Select Destination (Elevator System) ...................................................................................... 159

Figure 0.6: Class Diagram Generated by iSketch for Select Destination (Elevator System) ...................................................................................... 160

Figure 0.7: Sequence Diagram Generated by iSketch for Request Elevator (Elevator System) ..................................................................................... 161

Figure 0.8: Class Diagram Generated by iSketch for Request Elevator (Elevator System) ..................................................................................... 162
Figure 0.9: Sequence Diagram Generated by iSketch for Stop Elevator at Floor (Elevator System) ................................................................. 163
Figure 0.10: Class Diagram Generated by iSketch for Stop Elevator at Floor (Elevator System) ................................................................. 164
Figure 0.11: Sequence Diagram Generated by iSketch for Dispatch Elevator (Elevator System) ................................................................. 165
Figure 0.12: Class Diagram Generated by iSketch for Dispatch Elevator (Elevator System) ................................................................. 166
Figure 0.13: Sequence Diagram Generated by iSketch for ATM Client Validate Pin (Banking System) ............................................................ 167
Figure 0.14: Class Diagram Generated by iSketch for ATM Client Validate Pin (Banking System) ............................................................ 168
Figure 0.15: Sequence Diagram Generated by iSketch for Banking Service Validate Pin (Banking System) ................................................... 169
Figure 0.16: Class Diagram Generated by iSketch for Banking Service Validate Pin (Banking System) ................................................... 170
Figure 0.17: Sequence Diagram Generated by iSketch for ATM Client Withdraw Funds (Banking System) ..................................................... 171
Figure 0.18: Class Diagram Generated by iSketch for ATM Client Withdraw Funds (Banking System) ..................................................... 172
Figure 0.19: Sequence Diagram Generated by iSketch for Banking Service Withdraw Funds (Banking System) ....................................... 173
Figure 0.20: Class Diagram Generated by iSketch for Banking Service Withdraw Funds (Banking System) ....................................... 174
Figure 0.21: Benchmark Sequence Diagram for Process Cash Sale (NextGen Point of Sale System) ......................................................... 175
Figure 0.22: Benchmark Sequence Diagram for Process Credit Sale (NextGen Point of Sale System) ......................................................... 176
Figure 0.23: Benchmark Consolidated Class Diagram for NextGen Point of Sale System ................................................................. 177
Figure 0.24: Benchmark Sequence Diagram for Select Destination (Elevator System)........................................................................................................ 178

Figure 0.25: Benchmark Sequence Diagram for Request Elevator (Elevator System)........................................................................................................ 178

Figure 0.26: Benchmark Sequence Diagram for Stop Elevator at Floor (Elevator System)........................................................................................................ 179

Figure 0.27: Benchmark Sequence Diagram for Dispatch Elevator (Elevator System)........................................................................................................ 179

Figure 0.28: Benchmark Consolidated Class Diagram for Elevator System........................................................................................................ 180

Figure 0.29: Benchmark Sequence Diagram for ATM Client Validate Pin (Banking System)........................................................................................................ 181

Figure 0.30: Benchmark Sequence Diagram for Banking Validate Pin (Banking System)........................................................................................................ 182

Figure 0.31: Benchmark Sequence Diagram for ATM Client Withdraw Funds (Banking System)........................................................................................................ 183

Figure 0.32: Benchmark Sequence Diagram for Banking Service Withdraw Funds (Banking System)........................................................................................................ 184

Figure 0.33: Benchmark Consolidated Class Diagram for Banking System........................................................................................................ 185

Figure 0.34: Benchmark Consolidated Class Diagram Attributes for Banking System........................................................................................................ 186
List of Tables

Table 2.1 A Summary of Related Sketch Recognition Approaches ........... 39
Table 2.2 A Summary of Sketch Transformation Approaches .................. 43
Table 2.3 A Summary of Features Supported by Related Sketch Tools .... 49
Table 2.4 A Summary of Features to be supported by iSketch ............... 50
Table 4.1: The Architecture of iSketch CNN Model ............................ 69
Table 4.2 Grammatical Relations used in iSketch for Parsing NL Text ...... 81
Table 4.3 Syntactic Rules for Identification of Structures of NL texts used in iSketch ........................................................................ 82
Table 4.4 Properties of the Sketch-Based Model Transformation Method ......................................................................................... 92
Table 6.1: Mean Values for Recall, Precision and F-score of iSketch’s Sequence Diagram ...................................................................... 126
Table 6.2 Mean Values for Recall, Precision and F-score of iSketch’s Class Diagram .............................................................................. 128
Table 6.3 Comparison with Related Sketch Recognition Approaches ..... 130
Table 6.4 A Summary of Sketch Transformation Approaches .............. 132
Table 6.5 A Summary of Features Supported by Related Sketch Tools ... 134
Abstract

Requirements Engineering (RE) is the process of discovering and defining user requirements for developing software systems. The early phase of this process is concerned with eliciting and analysing requirements. Modelling, the activity of constructing abstract descriptions that are amenable to communication between different stakeholders plays a critical role in requirements elicitation and analysis.

However, current modelling tools are based on formal notations such as UML Diagrams and i* Diagram that do not support the use of hand-drawn diagrams or a mix of hand-drawn and computer drawn diagram to draw initial requirements models and subsequently transform the drawn models into target software models. The research presented in this thesis aims to address this problem. It aims to achieve two related objectives: 1) to develop a sketch tool, iSketch, that would enable users to draw use case diagram using either hand-drawn diagram or a mix of hand-drawn and computer drawn diagram. and 2) to support the transformation of the drawn use case diagram into initial software models represented by UML Class Diagram and UML Sequence Diagram.

Central to these research objectives are the development of novel sketch recognition and model transformation techniques for iSketch. To support sketch recognition, we have developed a deep learning technique that uses colour inversion to classify and improve the recognition rate of iSketch models. To support model transformation, we have developed a semantic modelling approach that works by first translating iSketch models into intermediate Agent-Oriented Models and finally into initial software models.

iSketch was evaluated in two ways. First, validation of iSketch through 2 experiments to measure the performance of iSketch in sketch recognition and model transformation using stroke labelling, and f-score metrics, respectively. In sketch recognition, iSketch achieved a recognition accuracy of 89.91% and 97.29% without and with colour inversion respectively when tested on iSketch dataset. In model transformation, iSketch achieved an f-score of 91.22% and 60.88% in generating UML Sequence and Class Diagrams respectively from iSketch models. Second, iSketch was compared with 15 related approaches. The result showed that only iSketch supports an automatic generation of initial software models from hand-drawn requirements models.
Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.
Copyright

i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii. The ownership of certain Copyright, patents, designs, trademarks, and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see http://www.campus.manchester.ac.uk/medialibrary/policies/intellectual-property.pdf), in any relevant Thesis restriction declarations deposited in the University Library, The University Library's regulations (see http://www.manchester.ac.uk/library/aboutus/regulations) and in The University's policy on presentation of Theses.
Acknowledgement

I am extremely thankful to my supervisor, Dr Liping Zhao, for her consistent support, guidance, advice, and interest in my research. I am immensely grateful for her constructive feedbacks over the years that have made me a far better researcher than I started.

I would like to thank all my friends at the University of Manchester, who had in one way or the other contributed to this work through their encouragements and discussions.

I owe a sincere gratitude to the Federal Government of Nigeria for the financial sponsorship received during my PhD.

My heart-felt gratitude goes to my parents for their love, support, and prayers all through the years.

To my lovely and special wife, Tolulope Olatunji, who was a pillar of support through my PhD, I am truly grateful for the peace of mind I enjoyed, the unwavering love, encouragement, kindness, and consistent prayers.

Above all, I thank God for the grace to be able to complete my PhD despite all odds. This would not have been possible without him.
Chapter 1 Introduction

1.1 Research Problem

Modelling is a fundamental activity in requirements engineering (RE), as it helps bridge the gap between the real world and the software. Through modelling, the requirements analyst creates an abstract description of the user requirements on a system and conveys this description to the software developer. Such a description is called a "model", which turns the user requirements into concepts and their inter-relationships.

There exist a variety of software tools for requirements modelling. These tools can be classified into three groups, namely: traditional tools, generic software diagramming tools and computer-aided software engineering (CASE) tools. Traditional tools are non-electronic tools that can be used for modelling requirements through hand-drawn diagrams only. They are the most dominant modelling tools used for exploring domain problems and hand drawing design solutions during the early stages of requirements elicitation [1], [2]. The reason for this is that they are flexible, easy to use and do not constrain the notations that can be used for modelling. However, there are disadvantages that prevent the traditional tools from being the perfect tool for requirements modelling. The most important drawbacks of these tools are their inability to 1) persist hand-drawn diagrams, 2) support manipulation of hand-drawn diagrams and 3) transform hand-drawn diagrams into initial software models. Consequently, users of these tools must manually re-model the hand-drawn diagrams from scratch using software modelling tools. This recreation process is tedious, time consuming and prone to errors. Examples of these tools include whiteboard, flipcharts, and paper and pencil.

Generic software diagramming tools are software tools that support creation of diagrams to model requirements mainly using mouse and keyboard. Like the traditional tools, the generic diagramming tools also do not constrain the notation that can be used for modelling [3] in that, users can use the tools to draw any type of diagrams to express
requirements models. In contrast to the traditional tools, they provide facilities that can be used for modifying and storing created models for a later reuse. These facilities help to address some of the drawbacks of the traditional tools. However, the generic diagramming tools do not allow users to express requirements models using hand-drawn diagram. Consequently, users are constrained to learning how to create various types of diagrams using the facilities provided by the tools. Also, the diagrams created by the tools are not formalized content and hence cannot be transformed into initial software models without a manual re-creation process using CASE tools. Examples of the generic software diagramming tools include LucidChart, Visio, and Draw IO etc.

CASE tools are software tools that support the use of one or more formalized pre-defined notations (e.g., UML Diagrams, i* Diagram etc.) to create diagrams for modelling requirements. CASE tools provide facilities that facilitate generation of initial software model, code, and documentation. They also give users the flexibility to store, edit and change diagrams [4]. However, CASE tools with their requirement to use pre-defined notations, focus on correctness and completeness. Users cannot freely create diagrams without being forced to use a specific notation or required to provide more detail than they would want [5]. Furthermore, users are constrained to learning specific modelling languages with formal syntaxes to be able to use CASE tools effectively [6]. Examples of CASE tools include UML CASE tools like Rationale Rose, Visual Paradigm, and Enterprise Architect etc.

The above description shows that there is a lack of support for less formal and user-friendly tools for end-users to hand-draw diagrams to convey their initial ideas of software requirements from the user perspective. Such tools are particularly valuable at the early stage of RE.

Researchers have recognized the need to overcome the challenge of providing a single requirement modelling tool that supports users during the early phase of requirements elicitation and have started tackling it from two directions.

First, by implementing a generic front-end to allow users express requirements through hand drawing of arbitrary diagrams. This led to various prototype tools such as SILK [1], InkKit [4], and Calico [7]. While these tools addressed the challenge of supporting user to flexibly hand draw diagrams and save them for a later re-use, they however do not support formalization and transformation of the drawn diagrams into initial
software models. Hence, the users still must re-create initial software models manually using CASE tools.

Second, by integrating sketch recognition facilities to allow users express requirements using hand-drawn diagrams and then subsequently beautify the diagrams into computer drawn diagrams. This resulted in various prototype tools such as Tahuti [8], SUMLOW [9] and OctoUML-Lite [3]. In addition to enabling beautification of hand drawn diagrams, one of the tools, FlexiSketch [10], supports users to semi-automatically generate a meta-model of their drawn diagrams. However, the shortcomings of these tools are twofold, 1) users are constrained to either using a single stroke or multiple strokes to hand-draw diagrams. Diagrams that do not adhere to the type of stroke expected by the tools cannot be interpreted by the sketch recognizer leading to errors during the beautification or transformation of the hand drawn diagrams. Hence, users are still limited in their expressiveness of requirements using hand-drawn diagrams [6], [11]. 2) Users are constrained to annotating elements in their diagrams with mainly names and types. They are not able to provide the text functional requirements of the software system being modelled, which can be used for documentation and automatic extraction of elements during initial software model generation from the diagram. Consequently, the users must manually re-create initial software models in CASE tools using their hand-drawn diagrams and the text functional requirements of the software system they modelled.

This research project is motivated to address the problem by developing a requirement modelling tool that: 1) does not restrict users to using a specific type of stroke (single or multiple strokes) for expressing requirements with hand drawing, and 2) enables users to automatically transform their drawings into initial software models. These problems will be addressed by utilizing Deep Neural Networks (DNNs) and PLANT’s Agent-Oriented Models (AOMs)

DNNs, especially convolutional neural networks have significantly improved and are promising for sketch recognition [12] while AOMs are suitable for identifying agents and their interactions needed for generating target initial software models from user requirements [13].
1.2 Research Aim and Objectives

The primary aim of this research is to investigate the feasibility of achieving a requirement modelling single user tool that enables users to express requirements using use case diagram with symbols drawn with any type of stroke (single or multi-stroke), in any stroke order and subsequently transform the diagram into initial software models. This aim is decomposed into the following specific objectives:

1. To investigate and compare different existing sketch recognition and transformation methods for processing hand-drawn requirements models.
2. To propose two novel methods that utilizes a) Convolutional Neural Network (CNN) for sketch recognition and b) Agent-Oriented Models (AOMs) for sketch transformation.
3. To implement and test the proposed methods through iSketch software proof-of-concept tool development.
4. To evaluate the proposed methods through benchmark datasets, test cases and comparison with the existing methods.

In line with these research objectives, the following research questions were defined:

RQ1. What sketch recognition and transformation methods are available for processing hand-drawn requirements models?
RQ2. What measures have been used to evaluate the performance of these methods? What are the limitations of these methods?
RQ3. Can deep learning and AOMs be used to address these limitations?
RQ4. How well can deep learning and AOMs support sketch recognition and model transformation?

1.3 Research Contributions

This research led to the following contributions:

1. A systematic study of existing sketch recognition and model transformation techniques and tools.
2. A Convolutional Neural Network (CNN) architecture that uses colour inversion information to address the problem of recognizing hand-drawn requirements model done with any type of stroke and stroke order.
3. A sketch-based transformation method that uses Agent-Oriented Models (AOMs) as an intermediate model for automatic construction of initial software models from hand-drawn requirements models to support the early stages of requirements engineering.

4. The development of a prototype software tool termed, iSketch that enables users to express requirements using hand drawings drawn with any type of stroke, in any stroke order and subsequently transform the diagram into initial software models.

1.4 Research Methodology

The research methodology used in my PhD project is depicted in Figure 1.4, which is adopted from Evidence-Based Software Engineering (EBSE) process [14]. This research consists of five steps, outlined as follows:

Figure 1.1: Research Methodology [14]

The first step entails identifying research problems by background work, justifying their significance, and converting them into answerable research questions. To ensure the validity of the identified research problems, various other disciplines should be explored and consulted.

The second step entails tracking down the best evidence with which to answer the identified research questions. To find best evidence, a literature search of relevant primary studies in leading publication venues should be done. This can be achieved by executing a search strategy against electronic databases of the various venues on the internet.
The third step entails a critical and detailed assessment of the validity, impact and applicability of the found evidence in practice. To critically appraise found evidence, a systematic review of identified primary studies should be carried out.

The fourth step involves addressing identified and justified research problems by presenting a new or an enhanced solution implemented in a software prototype. The implementation of a software prototype comprises the following:

- **Conceptual development** – entails investigating and identifying functional requirements needed to develop the prototype.
- **Architecture development** – entails identifying software components needed to implement identified functional requirements. Also, it involves defining the interaction between the identified system components.
- **Analysis/Design** – entails the design of data structures, entities, data transfer objects, databases and software architecture based on the identified software components. It also involves determining the required program modules, functions and classes that will be implemented to develop the prototype.
- **Implementation** – entails developing an executable computer program using the identified functional requirements and software components based on the analysis/design choices. During implementation, insights can be gained into the advantages and disadvantages of the used concept, architecture and chosen design. The insight gained is useful in re-designing the system if required.

The final step involves evaluating, testing the software prototype and seeking ways to improve it. The results of the evaluation are analysed based on the functional requirements of the prototype to establish the difference between the expected and actual output. The analysed results can determine a possible revision of the research problem or further development of the prototype. If any revisions are performed, another evaluation of the prototype will be carried out. The software prototype development cycle from conceptual framework identification to evaluation is repeated until the prototype can produce results that satisfy the research problems. If the research problems are satisfied and solutions provided by the prototype are validated, then the end results are research contributions.
1.5 Overview of Thesis

The remaining thesis is organized into the following chapters:

- **Chapter 2. Related Work.** This chapter details a literature review of related work in recognition and transformation of hand-drawn requirements models.

- **Chapter 3. Foundations for iSketch.** This chapter describes the background work that underpins my PhD project.

- **Chapter 4 iSketch Design.** This chapter introduces and describes recognition and model transformation method designed in this research project. It represents the main contribution of this research project.

- **Chapter 5 iSketch Implementation.** This chapter describes the research sketching software prototype tool implemented to validate the feasibility of the sketch recognition and model transformation method designed in this research project. It further describes its other features and their respective implementation. Finally, it demonstrates the process of using iSketch.

- **Chapter 6 Evaluation.** This chapter evaluates the performance of iSketch for sketch recognition and transformation. It further compares iSketch with related sketching tools.

- **Chapter 7 Conclusions.** This chapter summarises the results and contributions of the research project. It further discusses the limitations and challenges encountered in the current work. Finally, it proposes a direction for future work of this research.
Related Work

Chapter 2 Related Work

2.1 Introduction

Several attempts have been made to automatically process and recognize hand-drawn diagrams used for expressing requirements models. Hand-drawn diagrams may be referred to as requirements diagrams or sketches modelled using geometric primitives or UML elements and represented using UML notations. In addition to the automatic recognition attempts, some researchers have also tried to achieve a semi-automatic transformation of hand-drawn requirements into initial software models. With respect to these attempts, this chapter aims to introduce the state-of-the-art approaches and tools related to sketch recognition, transformation, and software tools in the context of hand-drawn requirements modelling during the early phase of requirements elicitation.

First, this chapter discusses related sketch recognition approaches for processing hand-drawn requirements model for recognition. It further presents a summary that can be used as taxonomy for sketch recognition approaches based on sketch recognition method and measures used for evaluating recognition performance. The taxonomy can serve as a guide for finding appropriate approaches in terms of how sketches are processed and the appropriate measures that can be used for evaluating recognition performance. Such taxonomy can help in understanding the domain of recognition of hand-drawn requirements models through the grouping of their respective advantages and limitations. Further, the taxonomy can serve as reference that can be used in deciding the suitability of a sketch recognition method for a particular recognition of hand-drawn requirements models.

Second, the chapter discusses existing sketch transformation approaches for generating a target model from hand-drawn requirements models.
Related Work

Third, the chapter presents sketching software tools that enable end users to express requirements models using either informal (hand-drawn) diagram notations, formal (computer-drawn) diagram notations or a mix of both.

Lastly, the chapter concludes with a summary of the related approaches.

2.2 Sketch Recognition Approaches

This section presents 20 identified related sketch recognition approaches for processing hand-drawn diagrams or sketches for recognition. Based on their intended purpose, these approaches can be classified into 4 broad categories:

- Gesture-based sketch recognition approaches
- Stroke-based sketch recognition approaches
- Geometry-based sketch recognition approaches
- Image-based sketch recognition approaches

2.2.1 Gesture-based Sketch Recognition Approaches

Related approaches in this category use gesture-based sketch recognition method to process hand-drawn requirements model for sketch recognition. The primary goal of the gesture-based sketch recognition method is to identify human gestures for expressing hand-drawing requirements model and use them to convey information. A gesture may be defined as a physical movement of the hands, arms, face and body with the intent to convey information [15]. The recognition done by this method tracks human movement and performs the interpretation of the movement as semantically meaningful commands that can be used to classify input gesture into a target output class. The general framework [16] used by the gesture-based sketch recognition method comprises: 1) storage of reference recognizable gestures, 2) acquisition of input gesture via frames from a webcam, 3) segmentation of the input gesture during which each frame is processed and analysed, 4) pattern recognition through the use of Hausdorff distance [17] to compare the segmented input gesture with the stored reference gestures and 5) carrying out corresponding action based on the recognized input gesture. Action such as to draw a specific symbol that represent the requirements model drawn through gesture.

The advantages of gesture-based sketch recognition method include providing users with gesture-based interfaces that do not constraint users to using traditional buttons.
and menus, additional devices such as keyboard and mouse. While users have flexibility through the use of gestures in expressing requirements models, the gesture-based sketch recognition method raises a major challenge that constraints users to either learn and remember how to draw a requirements model in the style required by the recognition method or train the recognition method with a particular drawing style [18]. Only one related approach that supports the use of gesture to sketch and process requirements model for recognition was found in the literature and it is described in what follows:

SkApp [19] integrated an online gesture recognition method for processing symbolic multi-touch gestures. The application allows users to specify their own sketches which consist of variable collection of strokes, shapes and relations to other sketches. Sketch variability is produced by specifying multi-touch, multi-stroke gestures per primitive or shape. The sketch recognition method used by the tool assumes gesture specification via a general top-down and left-right drawing direction and is capable of processing common primitives and their corresponding gestures for expressing UML class diagrams and designing graphical user interfaces while allowing multi-touch input.

Stroke labelling (SL) metric was used as a measure for evaluating sketch processing and recognition performance. The application achieved a recognition rate of 92% for UML class diagrams primitives and 88.5% for graphical user interface elements. However, the number of primitives and gestures used for calculating the recognition rate was not reported.

### 2.2.2 Stroke-based Sketch Recognition Approaches

Related approaches in this category make use of stroke-based sketch recognition method to process hand-drawn requirements model for sketch recognition. Stroke-based sketch recognition method is based on: 1) extraction of stroke structural information [20], [21] from an input sketch data. The extraction phase requires some pre-processing steps such as smoothing to remove noise from stroke points that represent the input sketch data, removal of duplicate stroke points, re-sampling of pixel points that form the stroke points, normalizing the stroke points into a fixed number of points and scaling to a pre-defined windows size. 2) Representation of the extracted information by a string, relational graph, unordered stroke set etc. 3) Use of dynamic programming or genetic algorithm or relaxation matching or linear programming to match the input sketch data with a target output symbol class. The recognition problems addressed by the stroke-
Related Work

Based sketch recognition method can be formulated as optimization problems using the following steps [22]:
a) identify the solution structure of the optimization problem e.g., stroke structural information of an input sketch,
b) code the solution in the form of strings,
c) define the objective function to be optimized,
d) define the operators and stopping criteria,
e) evolve solution until the stopping criteria are met and
f) decode the evolved solution strings into the optimum problem.

Stroke-based sketch recognition method depends on the stroke information of a sketch or hand-drawn diagram which can vary from user to user. The possible sketch variations comprise stroke order deviations, stroke number deviations, connected stroke deviations, incorrect stroke deviations and hybrid deviations. Stroke number deviations are as a result of missing or superfluous strokes, in some cases connected strokes or incorrect strokes [23]. The major shortcoming of the stroke-based sketch recognition method is the freedom to these deviations that does not constraint a user to using a particular stroke style to sketch requirements model while still achieving a high sketch recognition performance. Related approaches in this category are described in what follows:

Bresler et al. [24] introduced an online stroke-based sketch recognition method that understands diagrams such as flowcharts and finite automata. The sketch recognition method is implemented based on a recognition pipeline that performs:
1) text and non-text separation. This process separates possible text annotation used in a diagram to be processed and the symbols or shapes used for expressing the diagram.
2) Symbol segmentation where the strokes that constitute a diagram are broken down into subsets, each forming a particular symbol.
3) Symbol recognition which classifies the subset of strokes obtained during symbol segmentation into a symbol class. If a classification fails, the subset of strokes is rejected.
4) Structural analysis to detect a subset of candidate that forms a valid diagram using the candidates score and relations between candidates.

Sketch recognition performance evaluation was done using two metrics. The metrics are:
1) stroke labelling (SL) and
2) symbol segmentation and classification. Evaluation was done using three databases FC_A and FC_B for flow chart and FA for finite automata diagram elements recognition performance. With respect to the database FC_A, they achieved a stroke labelling recognition rate of 96.3% and symbol segmenta-
Related Work

tion and classification recognition rate of 84.2% on arrow, connection, data, decision, process, terminator and text diagram elements. With respect to the database FC_B, they achieved a stroke labelling recognition rate of 98.4% and symbol segmentation and classification recognition rate of 95.3% on arrow, connection, data, decision, process, terminator, and text diagram elements. With respect to the database FA, they achieved a stroke labelling recognition rate of 98.5% and symbol segmentation and classification recognition rate of 99.0% on arrow, initial arrow, final state, and label diagram elements.

Costagliola et al. [25] proposed a sketching framework that uses local context to enable the processing and recognition of hand-drawn diagrams. The framework has a layered architecture that comprises four layers: text/symbol separation layer, symbol recognition layer, local context detection layer and diagram recognition/parsing layer. The text/graphic separation layer separates hand drawings (symbols) from handwriting (texts). The symbol recognition layer recognizes and classifies drawn symbols into a symbol class. The layer achieves the recognition via stroke pre-processing, symbol identification and recognition. The local context detection layer identifies the attributes and their types for each symbol and the diagram recognition/parsing layer takes as input attribute-based representation and a visual grammar for syntax representation.

Although the approach was demonstrated on a simple example, no recognition performance evaluation was reported.

Deufemia et al. [26] presented and evaluated a grammar-based strategy for multi-domain hand-drawn diagram recognition. Sketch processing and recognition using the grammar-based strategy entails specifying a grammar for each symbols of a domain language and implementing a sketch recognition based on defined grammars. The recognition process consists of three phases. First the strokes that make up a diagram are interpreted as primitive objects such as lines, arcs, and ellipses etc. Second, the primitive objects are clustered to identify possible domain candidate symbols. Finally, recognition of the candidate symbols is carried out using a defined grammar and pruning of some symbol interpretations according to the recognition context.

Sketch recognition performance evaluation of the approach was carried out using precision and recall metric as measures. The sketch recognition performance achieved a precision rate of 99%, 94%, 65%, 68%, 91% and 92% for class, package, association,
aggregation, composition, and inheritance elements of a UML class diagram, respectively. The recall rates achieved are 95%, 98%, 84%, 85%, 80% and 89% for class, package, association, aggregation, composition, and inheritance elements of a UML class diagram, respectively. The evaluation was carried on 5 – 17 symbols.

Hammond et al. [27] presented a multi-stroke algorithm for processing hand-drawn diagrams. The multi-stroke algorithm has five parts: graph building, graph searching, stroke merging, false positive removal, and arrow detection. The first step of the multi-stroke algorithm constructs graph of spatially close strokes based on the assumption that candidate strokes that make up a multi-stroke primitive have endpoints that are likely near one another. Once the graph is constructed, a search for strongly connected components/sub-graphs is carried out. The identified strongly connected components indicate candidate strokes that need to be merged into multi-stroke primitives. Thereafter, stroke merging of indicated candidate strokes takes place. This process is significant to the approach’s recognition process. Once a stroke has been generated from the merging of other strokes, false positive removal is carried out to determine if the classification of that stroke is better than the result of leaving the strokes unmerged. Finally, arrow detection takes place to check if an addition stroke merging should take place.

Sketch recognition performance evaluation of the approach was carried out using stroke labelling metric. The approach achieved a weighted recognition rate of 96.0% on 15 primitive shapes.

SUMLOW [9] is an electronic whiteboard for end users to express early phase design of software in any UML models. The tool supports a progressive recognition of hand-drawn UML constructs and their formalization into computer-drawn notations with minimal interruption during sketching. The tool recognises a set of strokes as a UML element or Text. During the sketching or hand-drawing of diagrams, SUMLOW records the stroke’s pen movements used as a set of points. At each pen up, the tool either creates a sketch element with the new stroke or attempts to recognise the new stroke when added to a previous sketch element. SUMLOW then tries to recognise the sketch element as either a text or UML element (using a multi-stroke recognition method). The multi-stroke recognition method first takes a multiple stroke as input and categorises each of the strokes into a simple geometric shape. Subsequently, a second level
Related Work

recognition filtering is applied to: 1) recognize strokes within the geometric shapes and 2) carry out sizing comparison which, checks if adjacent shapes are correctly proportioned. The second level filtering is implemented based on two defined drawing rules: 1) the order and the number to complete and identifying stroke(s) of a UML symbol and 2), the relative position of strokes and the size of the sketch for a UML symbol. The two phases of the recognition process used by the tool allows it to combine shapes that can be recognised as UML notational symbols or not, through some developed drawing rules. The drawing rules contain describe how UML symbols must be drawn by the user. The rules define the order, number of strokes, relative position of the strokes and the size of the sketch used for a UML symbol. For example, an Actor UML symbol must be sketched starting from a circle (the head), a horizontal line (the arms), a vertical line (the body) and finally legs drawn with one or two lines.

The tool used stroke labelling metric to evaluate its sketch recognition performance. The tool achieved an average of 84.3% recognition rate in classifying UML symbols into 14 UML class diagram symbol classes.

Casella et al. [28] presented an agent-based framework for processing and interpretation of sketched symbol. The framework comprises four kinds of agents, namely: interface agent, input pre-processing agent, symbol recognition agents and sketch interpretation agent. The interface agent informs the sketch interpretation agent and the input pre-processing agent about the nature of the recognition process to use for sketched symbol processing. The input pre-processing agent segments and classifies users’ strokes into a sequence of domain independent primitives. The symbol recognition agents comprise of multiple agents, each devoted to recognizing a particular symbol of a domain. Each of the symbol recognition recognizes a domain symbol by applying applicable hand-drawn symbol recognizers to stroke classification provided by the input pre-processing agent. Finally, the sketch interpretation agent provides the correct interpretation of the sketch drawn so far or the entire sketch by resolving conflicts that may have occurred during recognition. The hand-drawn symbol recognizers works by: 1) identifying a sketched symbol stroke order or direction via generic framework, LADDER [29] that implements a rule-based sketch recognition method [30], 2) clustering identified strokes into symbols used in a domain language through the integration of Sketch Grammars [31],[32] and 3) processing the clustered strokes for recognition based on a Bayesian classifier [33].
The agent-based framework was evaluated using stroke labelling metric as a measure for its sketch recognition performance. On average, the framework achieved a recognition rate of 89.87\% in classifying UML use case diagram elements.

Brieler et al. [34] presented an approach for processing and recognition of hand-drawn diagram based on a syntactic and semantic analysis of an input diagram. The proposed approach performs sketch recognition and processing using the following steps: 1) it passes users’ strokes to a number of transformers, each processing the strokes according to their own criteria, and applying the resulting information into its corresponding model. 2) A recognizer is used to identify all components in an input diagram based on the information contained in the models and 3) identified components are passed to a generic diagram editor, DIAGEN [35] to present the recognition result to the user.

The approach was evaluated using processing time as a metric to measure how long it takes to analyse a hand-drawn diagram. The approach achieved 360ms in processing and recognising 39 components of a Petri net diagram.

Yuan and Jin [36] presented an intelligent whiteboard system based on sketch recognition. The sketch recognition algorithm of the system uses four steps to process and recognize a sketched symbol, namely: pre-processing of sketch stroke information, detection of feature points, recognition or primary symbols and classification of the symbol. During the pre-processing of a sketch, the recognition algorithm extracts the discrete coordinate data points of the sketch and then represent the data points as a tuple \((x, y, t)\), where \(x\) is the position of a data point on the x-axes, \(y\) is the position on the y-axis and \(t\) is the sampling time. The system applies a low pass filter to eliminate sampling error which in turn helps to extract the feature points of the sketch’s strokes. During the detection of feature points, the recognition algorithm extracts the minimum speed and maximum curvature of a sketch and then applies an intersection-based scanning algorithm to identify a candidate feature points. After the feature points are identified, the recognition algorithm applies linear least squares fitting for classification into a possible symbol class.

The sketch recognition performance of the system was not evaluated; however, an evaluation of the system was carried out via a usability evaluation by 160 computer science students. The usability evaluation objective was to find how pretty good, good, or
disappointing are the ease of use, efficiency, natural interaction, reliability, and users’ satisfaction of the system. The ease of use was evaluated to be 74.4% pretty good, 20.6% good and 5% disappointing. Efficiency was 84.4% pretty good, 14.4% good and 1.2% disappointing. Natural interaction was 93.8% pretty good, 6.2% good and 0% disappointing. Reliability was 57.5% pretty good, 26.3% good and 16.2% disappointing. Users’ satisfaction was 78.1% pretty good, 18.8% good and 3.1% disappointing.

*Tahuti* [8] allows end users to create UML class diagrams by using either a tablet or an electronic whiteboard. End users can draw and modify models while using a dual view, which allows users to view their original informal strokes and the automated created version of their strokes. The tool uses a multilayer framework for sketch recognition. The multilayer framework comprises pre-processing, selection, recognition, and identification of sketches. The pre-processing stage processes strokes only once, immediately after having been drawn. Strokes are fitted to fit an ellipse, a line, a polyline, or a complex shape. After a stroke has been pre-processed, the stroke is combined with unrecognized strokes to form a collection of strokes. The combined stroke is then sent to the recognizer for classification as a recognizable object or an editing command. Testing all the combinations of stroke for recognition would take exponential time which will be unacceptable for large diagrams. To address this challenge, the selection phase reduces the number of stroke collection for recognition via spatial, temporal rules and limiting number of strokes in a collection to a threshold. During the recognition stage, all stroke collections are tested for a possible classification as a viewable object or an editing command. The recognition algorithm used is based mainly on rectangle, ellipse, arrow and editing action recognition. If more than one interpretation is found for a collection of strokes, the final classification is deferred to the identification stage. During identification, a final classification is chosen. The identification stage selects the final classification using rules based on object movement, provided existing interpretation, number of strokes and correctness probability.

The tool’s sketch recognition performance was not evaluated. However, a usability evaluation of the tool on ease of drawing and editing was done and in comparison, with CASE tools such as Paint and Rational Rose. The subjects of the usability evaluation were asked to rank the ease of drawing and editing on a continuous scale from zero to five (zero being the hardest and five being the easiest). The results of the ease of use
are 3.5, 2.5 and 2.1 while that of ease of editing are 4.2, 2.7 and 1.1 for Tahuti, Rationale Rose and Paint respectively.

Hse and Newton [37] presented a system for recognizing and beautifying sketched symbols. The system applies recognition, fragmentation, and beautification to take a sketched shape or symbol to its beatified version. Once a shape is drawn using one or multiple strokes, the strokes are passed for recognition. After which the shape is structurally decomposed into its primitive elements such as line segments and elliptical arcs. From the structural elements, geometric properties are computed to derive the adjustment parameter for appropriate beautification.

The recognition and beautification performance of the system was evaluated using stroke labelling metric and three classification techniques: support vector machines (SVM) [38], minimum mean distance (MMD) [39] and nearest neighbour (NN) [39]. Furthermore, the evaluation was done using two scenarios: user dependent and independent test. For the user dependent test, the system achieved an average performance recognition rate of 97.10% (SVM), 91.15% (MMD) and 93.10% (NN). While for the user independent test, the system achieved an average performance recognition rate of 96.70% (SVM), 92.80% (MMD) and 94.00% (NN).

Sezgin and Davis [40] proposed a recognition framework based on Hidden Markov Models (HMMs). The proposed framework treats sketching as an incremental process, a sketch as a sequence of strokes and captures strokes using a digitizer that preserves stroke drawing order. After each stroke is added by a user, the framework encodes the new scene as a sequence of observations. The framework further achieves recognition and segmentation by aligning a series of HMMs to the sequence of observations. Each HMM models the drawing order of a single class of objects.

The proposed framework was evaluated using stroke labelling and running time comparison to a baseline method as metrics to measure the performance of its sketch recognition. For a set of 10 objects, the framework achieved a performance rate of 96.5% with stroke labelling metric.

SketchGIS [41] a sketch-based graphics input system for conceptual design, which is mainly based on online graphic recognition and dynamic modelling. The online graphic recognition feature of the system is used to discover primitive shapes from user-
drawn strokes and show the regularized or completed shape on the system’s user interface. The dynamic user modelling is used to recognize and predict sketchy composite shapes before they are completely drawn for user adaptation. The process of graphic recognition starts with pre-processing and segmentation of raw strokes by the tool’s stroke recognizer. In addition, the stroke recognizer classifies the pre-processed and segmented strokes as line/arc segments. After segment classification, the shape recognizer of the tool goes on further to perform primitive shape and composite shape recognition. The primitive shape recognition classifies and transforms the segmented strokes into a line, an arc or an ellipse primitive while the composite recognition refines the position relationship between primitives to form a spatial-relation graph for pattern matching and classification.

Although the system was demonstrated using a framework, no evaluation was reported.

*Lank et al. [42]* presented an online recognition systems that recognises a subset of UML Class, Use Case and Sequence diagrams. The proposed recognition system consists of a domain independent kernel and a UML-specific component. The domain independent kernel comprises of a graphical user interface for capturing the strokes of a hand-drawn diagram, a domain-independent segmenter which performs grouping and refinement of strokes for diagram recognition, character recognizer and user interface for correction of recognition errors. The UML-specific component performs a domain specific recognition using two steps: 1) identification of UML glyphs. This categorises glyphs as UML glyphs or characters and 2) application of recognition algorithm to the identified UML glyphs.

The system was tested on annotation of UML Class and Use Case elements. However, no evaluation of recognition performance was carried out.

### 2.2.3 Geometry-based Sketch Recognition Approaches

Related approaches in this category use geometry-based sketch recognition method to process hand-drawn requirements model for sketch recognition. Geometry-based sketch recognition method allows users to draw shapes that represent requirements models as they would naturally, by processing the shapes through important geometric constraints that hold for a shape [43]. In geometry-based sketch recognition method, a
requirements model shape is described in terms of the primitives it is composed of and the constraints on those primitives. The method works in two steps. The first is a low-level processing, where primitives like lines, arcs or circles that make up a requirements model shape are processed from the strokes drawn by the user. The second step is a high-level processing, where the primitives are used to compose the complete requirements model shape [44]. The high-level requirements model shapes are processed using the combination of low-level strokes and the constraints that specify how the low-level strokes fit together. This achieved using a search plan usually a tree that searches for primitives of a requirements model one after the other until all of them are identified. The search process takes into consideration the specification of the primitive and the constraints between primitives.

The main advantage of the geometry-based sketch recognition method is the clear separation between the low-level and high-level sketch processing steps. This enables the high-level processing step to not consider from which strokes a primitive emerges. However, to support drawing freedom of high-level shapes, a recognition system that uses this type of sketch recognition method must examine every possible combination of low-level shapes as well as every permutation of the low-level shapes. This implies that the algorithm based on the geometry-based sketch recognition method would take exponential time for sketch processing and recognition, which will be impractical for any non-trivial requirements model shape [43]. Related approaches in this category are described as follows:

*Fahmy et al.* [45] proposed an interactive sketch recognition framework capable of aggregating strokes that belong to the same primitive together for recognition using trained classifiers. The recognition framework can recognize 10 primitives comprising lines, triangles, rectangles, squares, circles, ellipses, diamonds, arrows, arcs, and zigzag lines. The framework architecture comprises grouping, classification pre-processing, features extraction, classification, and system output. Grouping of strokes is achieved using Euclidean distance grouping where the distance between two strokes endpoints is divided by the average stroke length of the two strokes and compared with some threshold. Any two strokes that are within the grouping criteria is merged and sent for classification pre-processing. During classification pre-processing, the proposed approach calculates the shape closure ratio and categorises user’s strokes into one of three categories: close shape, open shape, or incomplete shape. At the feature extraction stage, a set
of geometric features such as shape, size, and orientation to be used for classification is calculated. During classification, a stroke is sent to an appropriate classifier based on its category. The output stage of the approach replaces the user’s sketch with a beautified computer drawn diagram.

The proposed approach was evaluated using stroke labelling metric to measure its implemented sketch recognition performance. The approach achieved an average of 92.31% recognition rate across three classifiers (SVM, RF and K-NN).

Brieler and Minas [44] proposes a concept and architecture for a generic geometry-based sketch processing and recognition. The basic idea behind recognition with the proposed approach is description of diagram components in terms of primitives. To achieve the idea, the approach predefined several types of primitives. For recognition, primitives are searched for according to the description of components. If a primitive is found, it is then passed on for analysis. The process of recognition by the proposed approach entails: 1) specification which is used to define a small set of components to consist of primitives and constraints on the respective primitives. 2) Model generation for generating models to represent a certain view on component strokes and interpret them only regarding its view. 3) Recognition where primitives of a component are searched one after the other until all of them are identified and assembled. 4) A final elimination of recognized duplicate components.

To evaluate the recognition performance of the proposed approach, a user study was conducted and the average number of components identified was used as a metric to measure recognition performance rate for 5 diagrams, Petri net, Nassi-Shneiderman diagrams (NSD), Logic gate, Tic-tac-toe and GUI builder. The approach achieved an average recognition rate of 88.56%.

Hammond and Davis [43] presented an indexing technique for sketch recognition that examines all possible subsets of shapes when attempting to recognise new shapes. The technique uses efficient indexing to keep recognition performance close to real-time through fast pruning of strokes. The proposed technique performs recognition using a geometric method that composes shapes hierarchically from their sub shapes while testing the geometric constraints between them. The indexing technique works first by indexing geometric properties of a shape for use in recognition based a vocabulary of geometric constraints with which shapes are described. Constraints such as orientation
dependent constraints, orientation independent constraints, composite and constraints are defined. Second, handling of signal noises and conceptual variations based on shape descriptions and finally, use of constraint tolerances to remove possible shapes from recognition results.

The indexing technique was evaluated using a stress test to measure: 1) the time it took for a new symbol to be recognised when there is a large amount of unrecognized strokes and 2) to measure the percentage of recognition time focuses on each part of recognition.

*PaleoSketch* [46] proposed a low-level recognition and beautification system that can recognise eight primitive shapes as well as combination of the primitives. Shapes such as Arc, Circle, Curve, Ellipse, Helix, Line, Polyline, Spiral and Complex shapes are supported. The system starts by take a single stroke as input and performing a series of pre-recognition calculations, after which the stroke is sent to various low-level shape recognizers. Each recognizer in turn, provides a feedback stating whether the recognizer passed or failed as well as a beautified shape object that best fits the input stroke. Once the recognizers for all the possible shapes are executed, the recognition results are then sent to a hierarchy function which provides the interpretation of shapes in the order of best fit.

The system was evaluated using stroke labelling metric as a measure for its sketch recognition performance. On average, the framework achieved a recognition rate of 98.56% in classifying eight primitive shapes.

*Yu and Cai* [47] presented a domain-independent system that recognize freehand sketches and represent them with a hierarchy of primitive shapes and simple objects. The system aims at generality and all of its operations are based on low-level geometric features. This makes it possible for the system to be integrated with other high-level and domain specific applications. The whole process of the system entails imprecise stroke approximation and a post process. Imprecise stroke approximation is performed immediately after each stroke is finished. At this stage, a given input stroke is approximated to one or a combination of primitive shapes. The approximation result is then displayed as a feedback without interrupting the user. Post-process takes place after the user finishes the whole drawing. At this stage, the recognition system analyses the primitives
obtained from stroke approximation and attempts to represent them as what the user intends.

A user study was conducted to measure the recognition performance of the system. The system achieved a recognition rate of 98% for 8 primitive shapes and 94% from arcs primitive shape alone.

2.2.4 Image-based Sketch Recognition Approaches

Related approaches in this category use image-based sketch recognition method to process hand-drawn requirements model for sketch recognition. Image-based sketch recognition method treats hand-drawn diagrams as a collection of pixels, from which feature vector is extracted and calculated to perform classification and recognition by a statistical machine learning algorithm. It is generally based on: 1) feature extraction [48] of relevant shape information contained in a sketch or hand-drawn diagram data, 2) representation of the extracted information in a set of features (also named features vector) and 3) selection of features in the feature vectors for use in classifiers to recognize the input sketch data with a target output symbol class [49]. The main goal of the feature extraction used by the image-based sketch recognition method is to extract a set of features, which maximizes the recognition rate performance with the least amount of elements and to generate similar feature set for variety of instance of the same symbol [50]. Features extracted by the image-based sketch recognition method can either have a strong or weak relevance. Features with strong relevance are those that cannot be removed from the extracted features vector without loss of recognition accuracy while features that are of weak relevance can sometimes contribute to recognition accuracy [49]. Feature extraction is an important step for any classification and aims to extract the relevant information that characterizes each class. This makes it easier for the classifier to classify between different class label output by looking at these features. The classification label output depends on the sketch recognition problem at hand. For example, classifying a sketch diagram as either a use case diagram or a UML class diagram. In this instance, the class labels would be use case diagram and UML class diagram.

The major challenge of image-based sketch recognition method is the selection of the features with strong relevance which, can be time consuming or lead to a poor recognition performance if some relevant features are excluded. However, the main
advantage is that the image-based method is independent of stroke-order, number, as well as the aspect ratio and variation of a sketch data. Hence, a recognition approach that uses image-based sketch recognition method can provide a recognition system that does not constrain users to using a particular stroke order or type to sketch or hand draw a requirement model. Only one related approach that supports the use of gesture to sketch and process requirements model for recognition was found in the literature and it is described in what follows:

*Arueleba et al.* [51] proposed an image-based sketch recognition approach for extracting and processing hand-drawn requirements diagrams represented by Finite Automata (FA) notation. The proposed approach takes as input a digitized hand-drawn FA grayscale image and performs the following pre-processes for classification: removal of noise from the image using a median filter, conversion of the image to a binary image using a threshold value, normalization of the image to a predefined size of 40 X 40 pixels and conversion of the normalized image to single pixel thickness using thinning. After image pre-processing, the approach applies histogram of oriented gradients technique [52] as feature descriptor for feature extraction.

While the proposed approach demonstrated how to pre-process and extracts features from user's strokes in an example, the recognition performance of the approach was not evaluated.

### 2.2.5 Summary

This section explored the related approaches for sketch recognition. The identified approaches are summarised in Table 2.1 below.

<table>
<thead>
<tr>
<th>Recognition Approach</th>
<th># of Approaches</th>
<th>Approach Name</th>
<th>Evaluation Measure</th>
<th>Findings/Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gesture-based</td>
<td>1</td>
<td>SkApp [19]</td>
<td>Stroke labelling</td>
<td>Achieved recognition rate of 92.00% for UML Class diagrams and 88.50% for graphical user interface elements</td>
</tr>
<tr>
<td>Stroke-based</td>
<td>13</td>
<td>Bresler et al. [24]</td>
<td>Stroke labelling, symbol segmentation and classification</td>
<td>Achieved 96.70% average stroke labelling and 92.83% average symbol segmentation and classification recognition rate for flowchart</td>
</tr>
</tbody>
</table>
Related Work

<table>
<thead>
<tr>
<th>Recognition Approach</th>
<th># of Approaches</th>
<th>Approach Name</th>
<th>Evaluation Measure</th>
<th>Findings/Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry-based</td>
<td>5</td>
<td>Fahmy et al. [45]</td>
<td>Stroke labelling</td>
<td>Achieved an average of 92.31% recognition rate across three classifiers Support Vector Machines, Random Forest, and K-Nearest Neighbour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Brierer and Minas [44]</td>
<td>Stroke labelling</td>
<td>Achieved an average recognition rate of 88.56% for 5 diagrams, Petri net, Nassi-Shneiderman, Logic gate, Tic-tac-toe, and GUI builder</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hammond and Davis [43]</td>
<td>Stress test</td>
<td>Achieved a total time of 15660 milliseconds to recognize 56 shapes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PaleoSketch [46]</td>
<td>Stroke labelling</td>
<td>Achieved a recognition rate of 98.56% in classifying eight primitive shapes comprising arc, circle, curve, ellipse, helix, line, polyline, and spiral shapes</td>
</tr>
</tbody>
</table>

Table 2.1 A Summary of Related Sketch Recognition Approaches (cont.)
Related Work

<table>
<thead>
<tr>
<th>Image-based</th>
<th>Yu and Cai [47]</th>
<th>Usability test</th>
<th>Achieved an average of 96.00% recognition rate for primitive shapes, polylines, and arcs</th>
</tr>
</thead>
</table>

2.3 Sketch Transformation Approaches

An area of research in MDE is concerned with model transformations, that is, the transformation between one or more source models to one or more target models using a set of transformation rules [53], [54]. This section presents 5 identified related sketch transformation approaches that implement transformation of hand-drawn requirements model into a target or software model during early RE. These approaches take hand-drawn requirements model as an input and produces one or more target or software models as outputs. The sketch transformation approaches are described as follows:

*FlexiSketch* [55], [56] uses a lightweight meta-modelling approach implemented in a prototype tool that supports the definition of types for symbols and links and the definition of cardinalities for link types to semi-automatically generate a meta-model from hand-drawn requirements models [57], [6]. The process of generating a meta-model by FlexiSketch is as follows: 1) sketching of requirements models on a modelling canvas by an end user, 2) assigning of types to sketched models via annotation. The annotations provide the basic structure of the meta-model generated by FlexiSketch. 3) Inference connection cardinalities between sketched models.

A usability study of the tool implemented approach was conducted to find out the patterns of sketching and language definition that emerge when modelers collaboratively define lightweight modelling languages and extent novice, and expert modelers can define lightweight modelling languages correctly and completely with the proposed approach.

*metaBup* [58] is a tool implemented approach for an automatic generation of graphical modelling environments from a domain specific language example provided by an end user. The process of using the tool requires two roles: The Domain expert, who provides graphical examples and validates the generated environment and the Modelling Expert who monitors the meta-model generation process. The process of generating meta-model with the approach is as follows: 1) the domain expert provides input examples that may represent complete models or fragments. 2) The examples are
Related Work

automatically parsed into models and represented textually, making explicit the existing objects, attributes, and relations in the examples with annotating of their graphical rendering. 3) The modelling expert edits the textual representation to set appropriate names to the derived relations or to trigger a refactoring of the meta-model generation process. 4) The modelling expert exports the generated meta-model to a suitable format or invokes an editor generator to visualize the generated meta-model. 5) Finally, the domain expert validates the editor.

A running example was used to demonstrate the meta-model generation by the tool. However, no evaluation was carried out for model transformation.

Scribbler [59] is a model transformation approach that uses digital whiteboards to transform hand sketches to formal models and vice versa. The approach does not depend on a pre-defined modelling language and can interactively learn new graphical syntax elements and map these elements to formal meta-model entities. The process of model transformation with scribbler entails: 1) drawing of requirements models by the end user, 2) automatic transformation of the drawn models into a meta-model based on well-defined notations, 3) detection of collision and inclusion and 4) mapping of drawn objects to concrete elements of the generated meta-model.

Scribbler was evaluated through prototyping and a usability study that evaluates its ease of use.

MLCBD [60] (Modelling Language Creation By Example) is a user-centred approach for creating Domain-Specific Modelling Languages (DSMLs) from a set of domain examples provided by an end-user in a semi-automated manner. The proposed approach focused on 1) identification of concrete syntax, 2) inducing abstract syntax in the form of a meta-model and 3) inferring static semantics from a set of domain model examples. To generate a DSML from end-user model, the proposed approach uses graph theory and meta-model principles. The process of creating DSML by MLCBD entails: 1) demonstration or sketching of a set of model examples by end users in a modelling canvas, 2) transformation of domain model examples into a set of graphs, 3) identification of concrete syntax of a visual DSML from the set of graphs in semi-automated manner which entails identification of candidate concrete syntax from a set of graph representations of the domain model examples and a subsequent review by the end-user who is a domain expert, 4) regeneration of graph representations with the names of
Related Work

identified concrete syntax, 5) generation of meta-model from inferred abstract syntax and 6) inferring static semantics from the set of domain model examples.

The approach was demonstrated using examples of how to generate a meta-model from end-user models. But its model transformation process was not evaluated.

Model Workbench [61] is a tool implemented method for deriving a concise meta-model from end users model examples. The method comprises two main parts: 1) a bottom-up part where for each found unique type, a separate meta-concept is created with all required attributes. After the creation of meta-concepts, language patterns are applied to obtain a valid meta-model as defined by a Linguistic Meta Model’s (LMM) semantics and 2) a conciseness part during which analysis of the generated meta-model is done to find constellations of concepts to which further language patterns can be applied.

The approach was described with examples of generating a meta-model from end-users’ model examples. However, no evaluation was carried out.

This section reviewed related approaches for transforming hand-drawn requirements models into a meta-model. A summary of the reviewed transformation approaches, their corresponding inputs and outputs, and evaluation measures is shown in Table 2.2.

Table 2.2 A Summary of Sketch Transformation Approaches

<table>
<thead>
<tr>
<th>Approach Name</th>
<th>Input</th>
<th>Outputs</th>
<th>Evaluation Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlexiSketch [55], [56]</td>
<td>Node-and-edge sketch models</td>
<td>• Beautified diagram  • Metamodel XML file</td>
<td>Prototyping and user study</td>
</tr>
<tr>
<td>metaBup [58]</td>
<td>Graphical models</td>
<td>• Metamodel</td>
<td>Prototyping</td>
</tr>
<tr>
<td>Scribbler [59]</td>
<td>Sketch models</td>
<td>• Metamodel</td>
<td>Prototyping</td>
</tr>
<tr>
<td>MLCBD [60]</td>
<td>Sketch models</td>
<td>• Metamodel</td>
<td>Examples</td>
</tr>
<tr>
<td>Model Workbench [61]</td>
<td>Graphical models</td>
<td>• Metamodel</td>
<td>Prototyping</td>
</tr>
</tbody>
</table>

2.4 Sketching Software Tools

This section presents 15 identified sketching software tools that allow end users to express models using either informal (hand-drawn) diagram notations, formal (computer-drawn) diagram notations or a mix of both. Based on their intended purpose, these tools can be divided into four broad categories:
Related Work

- Early requirements engineering (RE) tools
- General modelling tools
- User Interface design tools
- General sketching tools

2.4.1 Early Requirements Engineering (RE) Tools

Tools in this category enable end users to draw diagram-like sketches during requirements elicitation. These tools are described as follows.

*FlexiSketch* [55], [62], [63] supports informal sketching of requirements models and sketch annotation. It is designed for interactive use and allows the end user to incrementally transform the sketch model into a semi-formal model. The tool provides three working modes for the sketching and incremental transformation process: 1) the modelling mode for the end user to create and modify informal sketches, 2) the meta-modelling mode for sketch annotation and transformation of sketch model into a semi-formal model, and 3) the sketch recognition mode for assisted sketch annotation and beautification of end user’s sketches. FlexiSketch can be used as a single-user tool for android devices or a multi-user collaborative tool.

*Calico* [5], [7] is an electronic whiteboard system for creative and exploratory purposes of early software design. The tool offers several features that amplify natural behaviours that software designers exhibit when they sketch a design. These behaviours include frequent shifting of focus, creating sketches with the detail and notation necessary to help them reason and the use of mixed notations during exploration of ideas on the whiteboard. The tool allows end users to 1) create and modify sketches, 2) create scraps from any part of the tool’s canvas and 3) sketch model on grid of canvases with the ability to explore different aspects and alternatives of a model in different grid locations. Furthermore, the tool also keeps a log of design histories, which can provide useful insights into the process by which users create software design drawings.

*SUMLOW* [9] is also an electronic whiteboard for end users to express early phase design of software in any UML models. The tool supports a progressive recognition of hand-drawn UML constructs and their formalization into computer-drawn notations with minimal interruption during sketching. The tool allows end users to 1) create and modify sketches, 2) make use of secondary annotations such as colour etc. during
Related Work

sketching, and 3) export sketches to a third-party CASE tool. In addition, the tool also allows the preservation of hand-drawn diagrams and supports multi-user software design collaboration.

2.4.2 General Modelling Tools

Tools in this category allow end users to express models in UML notations using hand diagrams. These tools are described as follows.

OctoUML [3], [64] is a tool for constructing UML class diagrams. Users can mix and match hand drawing sketches with the formal notations supplied by the tool. OctoUML can run on a variety of devices, ranging from desktop computers to touchscreens and electronic whiteboards. Furthermore, the tool provides a selective recognition mechanism that transforms hand-drawn sketches into formalized computer-drawn model contents.

Donaldson et al. [65] propose a tool that supports creation of UML activity diagrams with a pen-based input device such as electronic whiteboard or graphic tablet. Through a context-based feature inference, this tool can recognize the components of hand-drawn UML activity diagrams and their corresponding textual annotations. End users can make use of the tool’s training facility to specify examples of different components of UML activity diagram to enhance the recognition process.

Lank et al. [42], [66] developed a tool that allows end users to sketch UML class diagrams. End users can use an electronic whiteboard, a tablet, or a mouse to sketch the diagram. This tool incorporates a domain-independent segmentation and character recognition facility for recognizing sketches and a set of editing features for correcting segmentation and recognition errors. To use the editing features, the user changes the UML recognition mode of the tool from drawing mode to one of two editing modes: segmentation editing, or glyph editing. Segmentation editing allows correction of segmentation errors while glyph editing allows correction of glyph (UML or character) recognition errors and deletion of the entire glyphs.

Knight [4] uses an electronic whiteboard as its input medium to support the creation of UML class diagrams. The tool operates in two modes: 1) the freehand drawing mode in which users can make arbitrary sketches and annotations and 2) the UML mode where end users can sketch UML class diagram. Knight’s sketch recognition only works
Related Work

in the UML mode, by interpreting pen strokes as gestures for creating sketches. Knight makes use of the combination of eager recognition [19] and compound gestures to recognize the gestures made with pen strokes. Knight’s eager recognition facility starts processing a gesture as soon as the end user begins drawing the gesture and finishes when enough of the gesture has been recognized distinctly. Its compound gestures facility combines results from the classification of individual stroke gestures into a distinct gesture.

Tahuti [8] allows end users to create UML class diagrams by using either a tablet or an electronic whiteboard. End users can draw and modify models while using a dual view, which allows users to view their original informal strokes and the automated created version of their strokes. The tool uses a multilayer framework for sketch recognition. The multilayer framework comprises processing, selection, recognition, and identification of sketches. The recognition stage in the tool’s multilayer framework makes use of different recognition algorithms for rectangle, ellipse, arrow and editing action such as object deletion and movement.

2.4.3 User Interface Design Tools

Tools in this category enable end users to design user interfaces of software using hand drawings. These tools are described as follows.

FreeForm [67] allows end users to use a pen-based interface on an electronic whiteboard for designing Visual Basic (VB) forms. The tool provides a sketch space where users can draw, write, and edit sketches and a storyboard view where users can view their sketches and add navigation links between sketches. FreeForm incorporates a recognition engine that classifies pen strokes and a rule base that combines, and maps classified pen strokes to VB input controls (e.g., drop down lists, text boxes, buttons). In addition, FreeForm also provides a VB form creator that allows end users to automatically generate a VB form that contains input controls from their sketches. FreeForm further provides a run mode that allows end users to view generated VB forms and navigate from one form to another.

SketchiXML [68] is a multi-platform and multi-agent tool that enables end users to sketch a user interface (UI) with different levels of details and supports different contexts of use. The tool allows end users to produce UI specifications in several program-
Related Work

Coding languages to foster reusability and avoid restriction to a particular integrated development environment (IDE). In addition, end users can enable sketch recognition at different times and receive real time advice on their sketches. SketchiXML provides this advice in two phases: 1) a post-sketching phase that provides a set of usability advice based on the UI drawn and 2) a real time mode that looks for possible patterns, or similarities with previously drawn UIs.

SILK [1], [69] allows end users to sketch UIs quickly by interactive use of an electronic pad and stylus. The tool provides four windows: 1) the sketch window where users draw interface screens, 2) the controls window for performing different housekeeping operations on the sketches as well as giving user feedback on SILK’s recognition inferences, 3) the storyboard window for pasting interface screens copied from the sketch window and 4) the finished window that transforms sketch interface screens to a more formal interface using real widgets.

2.4.4 General Sketching Tools

Tools in this category provide general support for hand drawings, described as follows.

MaramaSketch [70] supports generic sketches. It also supports transformation of informal sketched models into computer-drawn model content. End users can make use of MaramaSketch meta-tools [71] to open or create a new modelling project. Once a modelling project is created, MaramaSketch provides a sketching facility that allows the end users to sketch, modify and manage models. Depending on end user preferences, sketched models may be immediately recognized and converted to computer-drawn model content; recognized but not immediately converted; or converted on-demand by the user after a whole design has been sketched.

Brieler et al. [34] provided a generic approach to generating diagram editors for supporting hand drawings. The tool allows end users to sketch with little restrictions on the sketch components. In addition, the tool incorporates a recognition algorithm that makes use of the syntactic and semantic information of sketch components to resolve ambiguities that can occur during a recognition process.

SketchREAD [72] a sketch recognition system that can be applied to sketch-based interfaces in a variety of domains. SketchREAD is capable of understanding and parsing
hand-drawn, messy, two-dimensional diagrammatic sketches. It parses and interprets sketches as they are drawn as distinct objects in any domain of interest. SketchREAD dynamically constructs Bayesian networks to interpret user’s sketches. The Bayesian networks also help its recognition facility to recover from some simple interpretation errors, such as a line misclassified as an arc.

**GDE [73]** a graphic diagram editor that uses a fast and simple multi-stroke shape algorithm for recognizing hand drawn diagrams. GDE can recognize rectangles, ellipses, circles, triangles, diamonds, lines, and polylines (joined sequence of line segments) regardless of their sizes. The most important feature of GDE is its ability to recognize geometric shapes drawn with multiple strokes in any order. The recognition process by GDE is triggered by a time-out event, which is initialized every time the user’s pen is taken out of proximity of the drawing canvas. GDE resets the timer when the user’s pen retouches the drawing canvas.

### 2.4.5 A Set of Common Features Supported by Related Sketch Tools

Based on the tools’ descriptions in sections 2.4.1 to section 2.4.4, seven (7) common features supported by some or all the 15 tools were identified and are summarized in Table 2.3. These features are briefly described as follows.

**FT1. Hand Sketch.** This feature allows users to use a finger or a touch screen pen to draw a shape or diagram. All the 15 tools support this feature.

**FT2. Computer-Aided Sketch.** This feature transforms shapes or diagram hand-drawn by users into computer-drawn shapes. It also allows users to select different shapes from a repository and compose them into a diagram. Only four tools (FlexiSketch [55], [62], [63], SUMLOW [9] OctoUML [3], [64] and Knight [4]) support this feature.

**FT3. Multi-Platform Sketch.** This feature allows users to use any device of their preference to draw a shape or diagram [29]. Depending on the device used, users can use either a finger or mouse for sketching. Only two tools (OctoUML [3], [64] and SketchiXML [68]) support this feature.

**FT4. Collaborative Sketch.** This feature allows a group of users to model requirements concurrently. Collaborative sketching can either take place in a co-located or distributed setting [30], [31]. In a co-located setting, users model requirements within the same location or facility synchronously whereas, in
a distributed setting user work together regardless of their location. Only three tools (FlexiSketch [55], [62], [63], SUMLOW [9] and OctoUML [3], [64]) support a co-located collaborative sketch.

FT5. Sketch Annotation. This feature allows users to assign properties such as name, type, and relationships to drawn shapes. Only two tools (FlexiSketch [55], [62], [63] and SUMLOW [9]) support this feature.

FT6. Sketch Recognition. This feature allows a sketch tool to process drawn shapes to deduce the properties of the shape by comparing it with those existing in a repository using a stroke dependent or independent recognition method. All the tools except (Calico [5], [7]) support this feature.

FT7. Sketch Repository. This feature enables a sketch tool to provide archives that can be used by users to store and reuse shapes or diagrams. In addition, Sketch Repository also facilitates the sketch recognition process of a sketch tool by providing archived shapes as templates that can be used for matching and deducing the properties of a newly drawn shape. Most of the tools (FlexiSketch [55], [62], [63], Calico [5], [7], SUMLOW [9], OctoUML [3], [64], Donaldson et al. [65], Knight [4], SketchiXML [68], SILK [1], [69] and GDE [73]) support this feature.

Table 2.3 A Summary of Features Supported by Related Sketch Tools

<table>
<thead>
<tr>
<th>Category</th>
<th>Tool</th>
<th>Hand Sketch</th>
<th>Computer-Aided Sketch</th>
<th>Multi-Platform Sketch</th>
<th>Collaborative Sketch</th>
<th>Sketch Annotation</th>
<th>Sketch Recognition</th>
<th>Sketch Repository</th>
<th>Stroke Dependency Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Early RE Tools</td>
<td>FlexiSketch [55], [62]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single Stroke</td>
</tr>
<tr>
<td></td>
<td>Calico [5], [7]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not Applicable</td>
</tr>
<tr>
<td></td>
<td>SUMLOW [9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
<tr>
<td>General Modelling Tool</td>
<td>OctoUML [3], [64]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single Stroke</td>
</tr>
<tr>
<td></td>
<td>Donaldson et al. [65]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single Stroke</td>
</tr>
<tr>
<td></td>
<td>Lank et al. [42], [66]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
<tr>
<td></td>
<td>Tahuti [8]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single Stroke</td>
</tr>
<tr>
<td>User Interface</td>
<td>FreeForm [67]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>Single Stroke</td>
</tr>
<tr>
<td></td>
<td>SketchiXML</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
</tbody>
</table>
Related Work

<table>
<thead>
<tr>
<th>Tools</th>
<th>Feature</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SILK [1], [69]</td>
<td>✓</td>
<td>Single Stroke</td>
</tr>
<tr>
<td>MarasSketch [70]</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
<tr>
<td>Brieler et al. [34]</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
<tr>
<td>SketchREAD [72]</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
<tr>
<td>GDE [73]</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
</tbody>
</table>

2.4.6 Features to be supported for iSketch

Except for the Collaborative Sketch feature (FT4), the remaining six features identified from the existing sketch tools are essential features that will be supported by the prototype tool, iSketch implemented in this PhD project. Collaborative Sketch feature will not be supported mainly because the primary aim of this project is to demonstrate the feasibility of sketch-based requirements modelling, recognition and transformation in a single user tool. In addition to the remaining six features, the following two new features will be supported by iSketch:

FT8. *Incremental Model Transformation.* This research project considers model transformation to be an essential feature of sketch tools, as will allow users to incrementally sketch and annotate models. It will further enable users to automatically transform sketched models directly into conceptual models without having to re-create those manually using CASE tools [6].

FT9. *Use Case Specification.* This feature will allow users to specify use cases by means of a structured natural language facility. This feature, together with sketch drawing, will help end users to express requirements models and provide more details for each model [74].

Table 2.4 A Summary of Features to be supported by iSketch

<table>
<thead>
<tr>
<th>Feature No.</th>
<th>Feature</th>
<th>Feature Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT1</td>
<td>Hand Sketch</td>
<td>Will provide hand-draw facility for users to create and save their custom notations that can be used to express requirements models.</td>
</tr>
<tr>
<td>FT2</td>
<td>Computer Aided-Sketch</td>
<td>Will provide a set of predefined computer-drawn notations that can be selected by users to express requirements models.</td>
</tr>
</tbody>
</table>
Related Work

<table>
<thead>
<tr>
<th>FT3</th>
<th>Multi-Platform Sketch</th>
<th>Will provide a user interface for users that can be used to express requirements models using any device of their choice.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FT5</td>
<td>Sketch Annotation</td>
<td>Will provide annotation facilities that can be used by users to specify the name, type, attributes and relationships of requirements models.</td>
</tr>
<tr>
<td>FT6</td>
<td>Sketch Recognition</td>
<td>Will provide a sketch recognition facility that can be used to validate the syntax and semantics of custom hand-drawn sketches created by users before they are saved in the Sketch Repository.</td>
</tr>
<tr>
<td>FT7</td>
<td>Sketch Repository</td>
<td>Will provide archives for storing predefined notations, hand-drawn notations and created requirements models.</td>
</tr>
<tr>
<td>FT8</td>
<td>Use Case Specification</td>
<td>Will provide an annotation facility that can be used by users to document the functional and non-functional requirements of use cases in their requirements models using structured natural language.</td>
</tr>
<tr>
<td>FT9</td>
<td>Incremental Model Transformation</td>
<td>Will provide a model transformation facility that can be used by users to generate initial conceptual models automatically from their drawn and annotated requirements models.</td>
</tr>
</tbody>
</table>

2.5 Limitations of Related Work

The reviews of related work in sketch recognition and transformation shows the following problems: 1) dependencies on stroke information for sketch recognition and 2) lack of an automatic direct transformation of hand-drawn requirements models into initial software models, respectively.

Dependencies on stroke information of shapes or primitives that represent hand-drawn requirements models for sketch recognition place limitations on how a user can express requirements models. Not adhering to the type of stroke information expected could lead to sketch recognition errors. Consequently, users will have to remember the type of stroke to use (single or multi-strokes), the order in which to draw the strokes, the size of the diagram expected etc.

Lack of an automatic direct transformation of hand-drawn requirements models to initial software models still introduces a manual process of using CASE tools to recreate the software models. This process is time consuming and error prone [55].
Chapter 3  Foundations for iSketch

3.1 Overview of iSketch

The purpose of this chapter is to introduce iSketch and its underlying foundational concepts. iSketch supports expression of Use Case diagram requirement models using a mix of hand drawings or sketch and computer drawn notations, and a subsequent model transformation into initial software models represented by UML Sequence and Class diagrams during the early phase of RE. Its sketch recognition method uses a sketch-based sketch recognition approach that is based on sketch classification with CNN models. While its sketch transformation method uses a bottom-up or example-based approach to dynamically construct a DSL for its source model and adapts PLANT pattern language in a dynamic meta-model construction and model transformation. iSketch sketch recognition and transformation method will be discussed in detail in chapter 5.

Based on the 8 features identified for iSketch in Section 2.4.6, 11 architectural components were needed to fulfil each feature. The 11 components and their respective interactions shown in Figure 3.1 are briefly described as follows:

1. **User Interface** will use a framework and technologies that will enable access to and the use of iSketch from multiple platforms ranging from desktop, laptop, and tablets to mobile phones. This component will help in achieving FT3.

2. **Sketch Canvas** will provide a facility for: 1) expressing requirements models using hand-drawn notations, computer-drawn notations, or a mix of the notations and 2) creating new symbol, shape or primitives using hand-drawing. This component will help in achieving FT1, FT2 and FT3.

3. **Sketch Annotator** will provide a facility for annotating symbols, shapes or primitives used in expressing requirements models. It will also support addition or documentation of the functional requirements of appropriate requirements symbol. This component will help in achieving FT5 and FT8.
Figure 3.1: iSketch Architectural Design. It shows its main components and their respective interactions, Control Flow (solid arrows), Data Flow (dotted arrows).

4. **Sketch Repository** will provide facility for storing created symbols and requirements models for a later reuse. This component will help in achieving FT7.

5. **Sketch Recognition Engine** will provide a sketch classification system to auto-suggest the syntax of hand-drawn sketches. This component will help in achieving FT6 and FT9.

6. **Sketch Model Transformer** will provide a function that dynamically constructs the concrete syntax that drives the model transformation process. It will also initiate the sketch to initial software model transformation process. This component will help in achieving FT9.

7. **Sketch Model Parser** will provide a function that will be used in parsing expressed requirements models. Its purpose will be to extract various components of the expressed models in line with the dynamically generated concrete syntax. This component will help in achieving FT9.

8. **Sketch Transformation Engine** will provide a sketch to initial model transformation engine that will be used to aid an automatically transform expressed or hand-drawn requirements model into initial software models. This component will help in achieving FT9.
9. *Sketch Model Interpreter* will provide a function that generates an intermediate model by extracting and parsing the natural language texts used in users’ expressed requirements models. This component will help in achieving FT9.

10. *Software Model Generator* will help to instantiate target models from the dynamically constructed meta-models. It will also help in achieving FT9.

11. *Software Model Realizer* will help in rendering the instantiated target models based on a predefined DSL and meta-model. It will also help in achieving FT9.

The mapping between the iSketch architectural components and the 8 identified features is shown in Figure 3.2, using the “Twin Peaks” model [75].

![Twin Peaks](image)

Figure 3.2: Twin Peaks showing the Relationship between iSketch Functional Requirements and its Architectural Components

### 3.2 Sketch Recognition and Deep Neural Networks

A sketch or hand-drawn diagram is the abstract representation of expressing some types of ideas. It has been considered as one of the most flexible and natural interfaces
Foundations for iSketch

in the human computer interaction (HCI) field [76]. A sketch image can convey information that is difficult to describe using text or other forms without a tremendous effort. With the rapid increase of touch screens, sketching has become easy, we can now sketch on phones, tablets and even on wrist watches. Consequently, research on sketches has flourished in recent years, with a wide range of application being investigated. Applications such as sketch recognition [77], [78], [79], sketch-based image or 3D model retrieval [80], [81], [82] and forensic sketch analysis [83], [84], [85]. Examples of sketches are shown in Figure 3.3.

Figure 3.3: Examples of sketches (TU-Berlin [73])

Recognizing sketches has long been an area of interest for researchers. Research work on sketch recognition can be traced as far back as early 1990s [86], [87], [88]. Since then, a lot of research has been carried out on sketch recognition, but it still poses significant challenges (see Figure 3.4). This is due to several reasons, unlike photos: 1) sketches are highly iconic and abstract, 2) sketches lack colour and texture information, which may cause sketches that connote different meanings appear to be similar and 3) the same sketch can be drawn in totally different drawing styles leading to varied level of abstractions. A large scale study on 20,000 sketches involving daily objects categorised into 250 classes evaluated humans to have sketch recognition accuracy of 73.1% [77]. This suggests that the sketch recognition task is challenging for humans as well. Prior methods for sketch recognition mostly make use of handcraft feature extraction techniques such as stoke structural information of sketch shapes [20], [21], geometry
properties of stroke shapes [43], [44], gestures for sketch shapes [89] followed by feeding them to a classifier. However, existing handcrafted features are low-level and do not account for the unique traits of sketches [90], [76]. An example of the prior methods for sketch recognition is shown in Figure 3.5.

Figure 3.4: An example showing the challenges of sketch recognition [76]

Figure 3.5: A flow chart example of prior methods for sketch recognition

In recent years, deep neural networks (DNN), especially convolutional ones have significantly improved the performance of photo-based image recognition e.g., [12], [91], [92]. Unlike prior methods of sketch recognition that extract shallow handcrafted
features, Convolutional Neural Networks is an end-to-end model trained from image to class label and contains millions of parameters that are dynamically learned from data. The basis of CNNs takes a biological inspiration from the visual cortex. The visual cortex has small regions of cells that are sensitive to specific regions of a visual field. This idea was further expanded upon by Hubel and Wiesel [93] where they showed that some neuronal cells in the brain fired only in the presence of edges of a certain orientation. For example, some neurons responded when exposed to vertical edges while some when shown horizontal or diagonal edges. Furthermore, Hubel and Wiesel found out that all these neurons were organized in columns and that together, they were able to produce visual perceptions. This idea of specialized components inside of a system having specific tasks (the neuronal cells in the virtual cortex having specific characteristics) is one that machines use as well. The CNN structures are mostly designed for the input of images or speech signals that are 2D or 3D.

3.2.1 CNN Architecture

A typical CNN architecture comprises several convolutional layers, pooling layers and followed by fully connect layers (see Figure 3.6). The operation of convolution is typically denoted by

\[ s(t) = (x * w)(t) \]  

Equation 3-1

where \( x \) is referred to as the input and \( w \) is the kernel or filter. The output is referred to as the feature map. A CNN automatically learns the values of its filter kernels during the training phase. Through the application of the filter kernels, a CNN is able to successfully capture the spatial and temporal dependencies in an image while reducing the image into a form that is easier to process, without losing features which are crucial for getting good prediction.

![Figure 3.6: A CNN Architecture example for Sketch Classification](image)

Figure 3.6: A CNN Architecture example for Sketch Classification
Central to CNN is the convolutional area that gives the network its name. This layer performs the convolution operation. Convolutional layer is the first layer in CNN and its operation is to extract the high-level features such as edges, from an input image. The element responsible for carrying out the convolutional operation in the convolutional layer is the kernel or filter or feature detector. There are two types of result to the operation of the filter: one, reduction of convolved features’ dimensionality as compared to the input image and the other, increasing or leaving the convolved features’ dimensionality same as that of the input image. This is achieved by applying **Valid Padding** in case of the former or **Same Padding** in case of the later. How a filter convolves across and image depends on parameters such as filter size, stride and zero padding. The size of a filter with respect to the size of the image determines the features that it can detect. Stride describes how many pixels a filter will be translated horizontally. Padding is a margin of zero values placed around the input image. The depth of padding can be set so that the output from a convolutional layer does no become smaller in size after convolution. With convolutional layers, reduction in image size can become a problem since some areas are lost at every convolution. Padding helps to address this problem by cancelling dimensional reduction and maintaining the input dimension at each convolutional layer’s output. To increase the non-linearity in the convolutional neural network, a ReLU layer can be applied to the convolutional layer. This is important as images themselves are non-linear objects.

Pooling layer, like convolutional layer, is also responsible for reducing the spatial size of the convolved feature. Reduction in the spatial size helps decrease the computational power required to process the input image data, thereby reducing overfitting. In addition, it helps to extract the dominant features of an input image. Pooling are of two types, namely: Max and Average Pooling. Max Pooling extracts the maximum value of the image area covered by the kernel. It also functions as a noise suppressant by discarding noisy activations. Whereas Average Pooling extracts the average of all the values of the image area covered by the kernel and performs dimensionality reduction as a noise suppressing mechanism.

Fully Connected layer is a traditional Multi-Layer Perceptron that uses an activation function (such as softmax, SVM, etc.) in the output layer for classification. The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer. The output from the convolutional and pooling layers is
Foundations for iSketch

high level features of an input image. The function of the Fully Connected layer is to use these features for classifying the input image into various classes based on a training dataset. In addition to classification, Fully Connected layers also learn non-linear combinations of image high level features for a better classification.

3.2.2 CNN Optimizations

During the network training, various optimizations can be carried out to improve the network’s sketch classification performance. Optimizations such as weight updates, dropout, and image data augmentation.

Weight updates is achieved through a method called backpropagation, where the gradient of an error function is computed with respect to all network weights. An error function is the sum of the squared differences between the network’s output and the correct output. The following optimizations methods can be used to update a network’s filter weights:

- **Stochastic Gradient Descent (SGD)** is used with batch processing, where the gradient is averaged over and used for making network weight updates.
- **SGD with Momentum** reduces irregular variations in steep gradient directions.
- **Nesterov’s Accelerated Gradient (NAG)** evaluates the next position indicated by the momentum. This is called “lookahead” gradient. In practice, NAG converges faster than the SGD with momentum.
- **AdaGrad** scales the gradient inversely proportional to the square root of the previous gradients. This means the network will have a larger learning rate than in steep directions.
- **RMSProp** is like AdaGrad but it applies a decay parameter to the cached sum of previous gradient values. This prevents the learning rate from going to zero.
- **ADAM** combines the concept of momentum and AdaGrad.

Dropout is a technique for preventing overfitting of a network to the particular variations of the training set [94]. In Fully Connected layers, which are prone to overfitting, dropout, can be used were nodes in a layer will be removed with some defined probability. Nodes that are removed do not participate in training and are replaced in the network with their original weights after training. This helps to prevent the Fully Connected layer from overfitting to the training data set.
Image data augmentation provides an effective way of increasing the size of a data set while not degrading the quality of the data set. Training on the augmentation data set yields networks which perform better than those trained on an unaugmented data set. Data augmentation comprises scaling, flipping, and rotating an input image.

### 3.3 Model Transformation

Model-driven engineering (MDE) relies on models to describe a system to be and aims to develop, maintain, and evolve the system through model transformations.

Models are simplified representation (or an abstract description) of a part of the world called system [95], [96]. A model is useful if it helps to express a better understanding of the system. In RE context, during the early phases of requirements elicitation; a model is useful if it helps in conveying the requirements or actions of the system to be.

Model transformation is the automatic generation of a target model from a source model according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language [97].

A model transformation process is driven by the following [54]:

- **What needs to be transformed into what?** This addresses the software artifacts of a given model transformation. Software artifacts may be a software program or models. In both cases, a model transformation comprises transformations from a more abstract representation to a more concrete model.

- **Number of source and target models.** A model transformation may be applied to multiple source models and or multiple target models, an example is model transformation from a Use Case diagram and Natural Language text source models into a UML Class and Sequence diagram target models.

- **Technical Spaces.** A given model transformation may have its source and target models belonging to the same or different technical spaces [54]. A technical
Foundations for iSketch

space is a model management framework that contains concepts, tools, mechanisms, languages and formalisms that are associated to a particular technology and it is determined by the meta-model used. [98]. Examples of technical spaces include XML, MDA, abstract syntax trees, database technology or ontologies etc. A technical space provides the schema of the meta-metamodel that supports the language used in that space. For example, the world-wide web consortium (W3C) uses the XML technical space that provides XML schema as the meta-model to support languages such as HTML, XML and XSLT. Tools and techniques are required to bridge the technical spaces of the source and target models. An example of such techniques is providing model exporters and importers while performing the model transformation in the technical space of either the source or target model.

- **Endogenous or exogenous transformations.** Model transformation requires that the source and target models be expressed in a particular domain specific language (DSL) whose syntax and semantics is defined by a meta-model. Depending on the DSL of a source or target model, a model transformation can either be an endogenous or exogenous transformation. An endogenous transformation takes place between models expressed in the same DSL whereas exogenous transformation takes place between models expressed in different DSLs. Examples of endogenous transformation include optimizing the performance of a system while preserving its semantics and refactoring the internal structure of software to improve its non-functional characteristics such as reusability, modularity, and adaptability. Examples of exogenous transformation include transforming a higher-level or more abstract specification into a more concrete one and a reverse engineering of a more concrete specification into a more abstract one.

- **Horizontal or vertical transformations.** A horizontal transformation is a transformation where the source and target models are of the same abstraction level whereas a vertical transformation is one in which the source and target models are at different level of abstractions.

- **Syntactic or semantical transformations.** Syntactic transformations entails transforming a concrete syntax of a source model into an abstract syntax which can
then be used to form the internal representation of the source model on which a semantical transformation can be applied to generate a target model.

To verify the correctness of a given model transformation, an option is to prove the theoretical properties of the transformation such as termination, soundness, completeness (syntactic and semantic) or correctness. The simplest notion of correctness is syntactic correctness that evaluates if a target model is well-formed from a given well-formed source model. Another notion is syntactic completeness that evaluates if each element in the source model has a corresponding element in the target model created by a model transformation. A more complex notion is semantic correctness which evaluates if the produced target model has the expected semantic properties.

3.4 DSL and Metamodel Construction

Models are usually built using domain specific languages (DSLs) which are specified through a meta-model. A DSL should contain useful, appropriate primitives and abstractions for a particular application domain [99]. The meta-model of a DSL specifies all available concepts in the language and imposes restrictions how these concepts can be used [100].

3.4.1 DSL Construction

The usual process of DSL construction requires building a part of the meta-model through examples and then a subsequent building of instance models. This process is time consuming and requires an in-depth knowledge about meta-modelling and appropriate tools that have a steep learning curve [101]. The construction of a DSL is typically carried out by a language engineer that makes use of inputs from domain expert to design the appropriate formalism of the meta-model. The formalism of a DSL comprises the:

- **Abstract syntax** defines the structure of the concepts of a DSL. Abstract syntax is typically a data structure that describes the DSL concepts and their relations. This is usually done by the meta-model of the language.
- **Concrete syntax** defines how the DSL concepts are represented using either textual or graphical notations.
- **Semantics** define the meaning of the DSL concepts.
To construct a DSL from scratch, the most commonly used approach is a top-down approach [102] which entails: 1) gathering and analysing requirements of the DSL from the domain experts, 2) defining the abstract, concrete and semantics of the DSL based on the requirements by language engineers and 3) development and testing of the DSL. Figure 3.7 shows the activities within a top-down approach of DSL construction.

The top-down approach for DSL construction has one major shortcoming that has to do with its meta-model construction. The meta-model of a DSL describes its abstract syntax and is derived from the DSL requirements provided by domain experts. The challenge here is that the domain experts may not be aware of all the requirements at the initial phase. In practice, the domain experts often start with drafting example models
first which then refined into elements of the envisaged DSL [99]. Another shortcoming of the top-down approach of DSL construction is the separation of activities done by domain experts and language engineers. While domain experts have the required domain knowledge that can be used in constructing a DSL, they however most often do not have the required competence to design a meta-model for the language [103].

Figure 3.8: Example-driven approach for DSL construction adapted from [53]

A promising trend that addresses the challenges of the top-down approach for DSL construction is example-based meta-model construction. Through a set of example instance models, the example-based meta-model approach automatically infers a meta-model that the instance models conform to. The process of using the example-based meta-model approach comprises:

- A *brainstorming phase* during which domain experts express their ideas of a DSL using example models. Commonly, these example models are hand-drawn diagrams
- A *refinement phase* where the DSL’s meta-model is automatically generated from the example models. The generated meta-model can then be refined incrementally as modifications are made to the example models.
• A modelling phase that ensures conformance of the relationship between the meta-model and the example models.

Example-based meta-model approach can also be referred to as demonstration-based or bottom-up approach for DSL construction. Figure 3.8 depicts the phases of the example-driven approach for DSL construction.

3.4.2 Metamodel Construction

One important goal in meta-model construction is keeping meta-models concise [104]. Hence, models should be as small as possible and describe their respective domains with as few constructs as possible. Making a meta-model concise can be achieved through the use of language patterns [61]. A language pattern within the scope of this study for concise meta-model construction is Pattern LANguage for Transforming scenarios [13], [105]. Within RE, scenarios are example models that can be used as a communication tool for bridging the gap between technical and non-technical stakeholders during requirements elicitation [106], modelling [107] and specification [108]. The PLANT language defines a sequence of four patterns that be used to guide transformation of a certain part of example models into target models. The patterns are:

• Establishing the story line transforms the main story line of an example model into a process-orient model (POM) with structural elements that comprises states, events, decision points and their temporal relationships.

• Elaborating thing that change elaborates the dynamic part of example models into object transformation model (OTM). The structural elements of an OTM comprise object, states, state transitions and actions.

• Identifying agents and their interactions identifies the agents and their interactions of example models and represents them using agent-oriented model (AOM). The structural elements of an AOM comprise agents and actions.

• Unravelling the goal and its sub goals discovers the goal of an example model and decomposes them into goal-oriented model (GOM). The structural elements of a GOM comprise attainment goal and avoidance goal. The concepts of pattern and pattern language of PLANT are based on Alexander’s pattern language for defining building designs [109], [110]. The theoretical foundations of PLANT
are based on cognitive structures [111], [112] and requirements engineering derived meta-models [113], [108], [114], [115], [107].

Based on PLANT, a scenario or example model can be structured using a SOURCE-PATH-GOAL schema, where:

- SOURCE contains the initial state that defines the preconditions for triggering an example model
- PATH contains a flow of actions
- GOAL contains the final state that defines the outcome of triggering a scenario.

### 3.5 Summary

This chapter described the fundamental work that is used in this study. Investigation of sketch recognition and deep neural networks provided insight into a sketch recognition method that overcomes the challenge of low-level or shallow dependency on stroke information of shape, primitive or requirements model for sketch classification and recognition. It examined Convolutional Neural Network (CNN) architecture and possible optimizations that can be used for designing, implementing, and training a CNN model that uses the deep features of a sketch data for its classification and recognition. Sketch recognition enhances the sketching experience of users through an auto suggestion or completion of the users’ sketch.

Once a user has completed modelling the requirements of a system to be, there is a need for an initial software model to be generated to support MDE. The study on model transformation gave the characteristics of a given model transformation method and how to evaluate the transformation done by the method.

Finally, to support automatic model transformation, there is a need to understand how to construct the DSLs for the source and target models as well as their respective meta-models. A study of DSL and meta-model construction gave insight into how to achieve this. Consequently, the sketch recognition and transformation which are described in the next chapters build on these fundamental works.
Chapter 4  iSketch Design

4.1 Introduction

The purpose of this chapter is to present and discuss the sketch recognition and transformation method developed to address the shortcomings of existing related works.

As discussed in Chapter 2, the shortcoming identified for sketch recognition is dependency on: 1) stroke information such as stroke type (single or multiple), order and aspect ratio, 2) gestures and 3) geometry properties of strokes used for sketching or drawing a requirements model shape or primitives. These dependencies introduce constraints on how a user sketches or expresses requirements models. Constraints such as remembering what stroke type or gestures to use in expressing a requirements model. Any deviation from what type of stroke expected will yield a sketch recognition error. For sketch transformation, the shortcoming identified is a lack of an end to end automatic model transformation method for generating initial software models from sketched or hand-drawn requirements models. Users would still need to make use of CASE tools to manually recreate initial software models from scratch.

The sketch recognition and transformation method developed in this research attempts to address these shortcomings. The methods are termed Sketch-Based Recognition Method and Sketch-Based Model Transformation Method, respectively. They are discussed in what follows.

4.2 Sketch-Based Recognition Method

Figure 4.1 shows the framework of the proposed sketch-based recognition method. The proposed method uses CNN to develop a sketch recognition model that can be used for the recognition of unknown sketches.
The sketch recognition model developed in this research is termed iSketch CNN Model. The model used a deep CNN for feature extraction and sketch recognition. The development and training of iSketch CNN model entails developing: 1) an offline data augmentation pipeline and 2) iSketch CNN architecture.

### 4.2.1.1 Offline Data Augmentation

Offline data augmentation is performed by transforming a training data outside a neural network by means of image processing algorithms. In this research, four sketch transformations were done to increase the size of the sketch data used for training iSketch CNN model. The sketch transformations are as follows:

- **Flip/Mirror.** This flips a sketch data across horizontal, vertical and both symmetrical axes.
- **Rotate.** This rotates the original sketch as well as the flipped sketch by a random angle between -45 and 45 degrees.
- **Crop.** To optimize the training of iSketch CNN architecture on CPU computers, all sketch data were cropped to a size of 32 X 32.
- **Colour Inversion.** This inverts the colour of a sketch data such that data pixels of the sketch became white on a black background.
The data augmentation pipeline takes a sketch data through the sketch transformations and save it in readiness for iSketch CNN training. An example of the output of data augmentation implemented for sketch-based recognition method is shown in Figure 4.1.

Table 4.1: The Architecture of iSketch CNN Model

<table>
<thead>
<tr>
<th>Index</th>
<th>Layer</th>
<th>Type</th>
<th>Filter Size</th>
<th>Number of Filters</th>
<th>Stride</th>
<th>Padding</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Input</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>32 x 32</td>
</tr>
<tr>
<td>1</td>
<td>L1</td>
<td>Conv</td>
<td>3 x 3</td>
<td>32</td>
<td>1</td>
<td>Same</td>
<td>32 x 32</td>
</tr>
<tr>
<td>2</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>32 x 32</td>
</tr>
<tr>
<td>3</td>
<td>MaxPool</td>
<td>2 x 2</td>
<td>-</td>
<td>1</td>
<td>Same</td>
<td>-</td>
<td>16 x 16</td>
</tr>
<tr>
<td>4</td>
<td>Dropout (0.15)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>16 x 16</td>
</tr>
<tr>
<td>5</td>
<td>L2</td>
<td>Conv</td>
<td>3 x 3</td>
<td>64</td>
<td>1</td>
<td>Same</td>
<td>16 x 16</td>
</tr>
<tr>
<td>6</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>16 x 16</td>
</tr>
<tr>
<td>7</td>
<td>MaxPool</td>
<td>2 x 2</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>8 x 8</td>
</tr>
<tr>
<td>8</td>
<td>Dropout (0.20)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8 x 8</td>
</tr>
<tr>
<td>9</td>
<td>L3</td>
<td>Conv</td>
<td>3 x 3</td>
<td>128</td>
<td>1</td>
<td>Same</td>
<td>8 x 8</td>
</tr>
<tr>
<td>10</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>8 x 8</td>
</tr>
<tr>
<td>11</td>
<td>MaxPool</td>
<td>2 x 2</td>
<td>-</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>4 x 4</td>
</tr>
<tr>
<td>12</td>
<td>Dropout (0.25)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>4 x 4</td>
</tr>
<tr>
<td>13</td>
<td>L4</td>
<td>FC</td>
<td>1 x 1</td>
<td>1024</td>
<td>1</td>
<td>-</td>
<td>1 x 1</td>
</tr>
<tr>
<td>14</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1 x 1</td>
</tr>
<tr>
<td>15</td>
<td>Dropout (0.30)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1 x 1</td>
</tr>
<tr>
<td>16</td>
<td>L5</td>
<td>FC</td>
<td>1 x 1</td>
<td>4096</td>
<td>1</td>
<td>-</td>
<td>1 x 1</td>
</tr>
<tr>
<td>17</td>
<td>ReLU</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1 x 1</td>
</tr>
<tr>
<td>18</td>
<td>Dropout (0.35)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1 x 1</td>
</tr>
</tbody>
</table>

4.2.1.2 iSketch CNN Architecture

After developing the augmentation pipeline, iSketch CNN architecture was designed. The architecture which is summarised in Table 4.1 follows a typical structure of multiple convolutional layers followed by fully connected layers. It is described as follows. The size of the input sketch is 32 (sketch height) x 32 (sketch width) x 1 (sketch depth). Each convolutional layer is followed by 1) a rectifier layer (ReLU) that maintains the non-linearity of the input sketch, 2) maximum pooling layer (Maxpool) and 3) dropout. The dropout used in the network is increased gradually after each level. This helps the network to randomly drop sketch training data to prevent overfitting of the network to a particular variation of the training sketch set. Finally, 2 fully connected layers were added to the network with dropouts of 0.30 and 0.35, respectively. The out-
put layer has a few output units that is determined during training and uses a softman loss for sketch classification.

Most CNNs are proposed without explaining the reason why certain hyper parameters such as filter size, stride, and number of filters, padding and pooling size were chosen. Although it will be seemingly difficult to exhaustively verify the effect of every hyper parameter. The following discusses some points that are like classic design and those that are specifically designed for sketches and hence differentiates iSketch CNN architecture from image-based CNN architecture.

**Similarities between iSketch and Existing Image-Based CNN Architecture**

- **Filter Number:** In both iSketch and image-based CNN architecture such as [12], [92] and [116], the number of filters increases with the depth after every pooling layer in the network.
- **Stride:** Like image-based CNN architecture, the stride used in the convolutional layers is set to 1. This ensures that as much sketch information as possible is kept and not lost with the convolutional operation.

**Differences between iSketch and Existing Image-Based CNN Architecture**

- **Colour Inversion:** In contrast with image-based CNN architecture that do not make use of colour inversion during the pre-processing phase, this research found out that colour inversion data augmentation improves the sketch recognition accuracy of the sketch-based CNN model. For example, an experiment (described in chapter 6.2) using the architecture shown in Table 4.1 achieved sketch recognition accuracies of 98.75 % and 91.46 % when trained with colour inverted and non-inverted sketch data, respectively. Both networks were trained on 3,840 sketch data samples and validated on 960 sketch data samples to classify a sketch data into 6 classes. Although, the differences in the recognition accuracies seem small but as the number of classes for sketch classification increases, the differences will become more significant.
- **No Local Response Normalization (LRN):** LRN implements lateral inhibitions which in neurobiology, refers to the capacity of an excited neuron to subdue its neighbours. Lateral inhibitions create a contrast in formed local maxima that helps to increase sensory perception of an image. In image-based CNN, imple-
menting LRN helps to normalize ReLU neurons which have unbounded activations. Through the normalization, high frequency features with a large response can be detected and normalized around the local neighbourhood of the excited neuron. This makes the neuron more sensitive as compared to its neighbours. LRN basically boosts the neurons with relatively large activations. However, LRN is mainly due to providing contrast or brightness normalization. This is not necessary in sketches since contrast or brightness is not an issue in line-drawings. Hence, not using LRN layers achieves a faster network learning rate without sacrificing performance.

- **Lower Computational Cost:** The total number of parameters of iSketch CNN architecture is 7.4 million, which is relatively small for modern CNNs. For example ImageNet [12] has 60 million parameters and [116] has 144 million parameters.

### 4.3 Sketch-Based Model Transformation Method

The design of the sketch-based model transformation method entails: 1) developing a sketch-based modelling language and 2) developing a process flow and accompanying set of rules for model transformation.

#### 4.3.1 Sketch-Based Modelling Language

The sketch-based modelling language uses an example-based or demonstration-based approach in defining a domain specific language for sketch-based model transformation. The language provides key features like abstract syntax, concrete syntax, and semantics.

The abstract syntax of the sketch-based modelling language describes the vocabulary and the concepts provided by the language and how they can be combined to generate initial models from hand-drawn requirements models. It consists of a set of provided concepts, their respective relationships and rules that define whether the DSL is well-formed. In terms of MDD, the abstract syntax is described by a meta-model that defines how the model instantiated should look like. The abstract syntax of the sketch-based modelling language is defined using an adaptation of PLANT models [13], [105]. The concrete syntax of the language provides a set of notations that facilitates the presentation and dynamic construction of the language constructs during an automatic
model transformation process. The semantics of the language provides a validation of the syntax of the language. It helps to ensure a correct definition of the language’s concepts and prevent assumption that can lead to an incorrect usage of the language.

Figure 4.2: Abstract Syntax of iSketch Sketch-Based Modelling Language

Three types of models provide the main building blocks for the abstract syntax of the sketch-based modelling language. They are: 1) source, 2) intermediate and 3) target model.

**Source Model** describes the requirements model expressed by the user. In this research, Use Case diagram was used as a notation for expressing requirements models. The source model comprises actors and use cases of the system under design. It is based on the Process-Oriented Model and Object-Transformation Model of PLANT. With the source model, users can specify the actors of a system to be, and their respective attributes such as name and properties (e.g. height, registration number, email etc.). Furthermore, users can specify the roles that the actors can play in the system to be via the use of use cases. In addition to specifying actors and their use cases, the source model also allows users to specify the functional requirement of the use cases using a structured natural language text that provides placeholders for specifying the description, success scenarios, preconditions and success guarantees of use cases.

**Intermediate Model** describes the entities and their respective properties that can act in the system under design. Through the process of source model transformation, agents that act as intermediate models are generated. The target model is based on the Agent-Oriented Model of PLANT. It allows a dynamic specification of agents and their interaction based on the source model using an AGENT-INTERACT-AGENT schema. It is central to the model transformation process of the sketch-based model transfor-
iSketch Design

Information method and contains three concepts, namely: Source Agent, Target Agent, and Interaction Message.

Target Model represents the initial software models that are generated from the intermediate models. In this research, Class and Sequence diagrams are used as notations to represent initial software models. The target model is based on the Goal-Oriented Model of PLANT. It facilitates the generation of initial software models, their respective properties, relationships, and interactions from the intermediate models.

![Stringified and parsed JSON](image)

Figure 4.3: Concrete Syntax of iSketch Sketch-Based Modelling Language

The concrete syntax of the sketch-based modelling language is specified using the JavaScript Object Notation (JSON) specification [117]. JSON can represent four primi-
tive types (strings, numbers, booleans and null) and two structured types (objects and arrays). To construct the concrete syntax, the building blocks of the abstract syntax were mapped to appropriate primitive and structure types supported by JSON. The mapping is as follows:

- **Actors.** Mapping of an actor to an object structured type with name/value pairs of members. The members of an actor comprise the name and attributes primitives, association, and generalization object structured types. Association and generalization define the relationships an actor can have with a use case or another actor, respectively. Actors are then mapped to an array structured type that contains a list of individual actors.

- **Use Cases.** Mapping of a use case to an object structured type with name/value pairs of members. The members of a use case consist of the description, success scenarios, preconditions and success guarantee primitive types, include, extend and association structured types. Use cases are then represented using an array structured type with elements that are of the use case object.

- **Agents.** Mapping of an agent to an object structured type that contains members defined by name/value pairs. The members of an agent comprise the source agent, target agent and interaction message primitives. Agents are then represented as an array of individual agents.

- **Source Model.** Mapping to an array structured type. Its elements are arrays of actors and use cases.

- **Intermediate Model.** Mapping to an array structured type. Its element is an array of agents.

- **Target Model.** Mapping to an array structured type and contains an array of targets.

Figure 4.3 depicts the concrete syntax of the sketch-based modelling language

The abstract and concrete syntax of a language mainly contains little information about what the concepts in a language means, describes the vocabulary of the concepts and how they may be combined or represented to create models. Hence, there is a need to use additional information and mechanisms to capture the semantics of the language. Consequently, a complete formal specification was developed using Alloy [118]. Alloy is a declarative specification language for expressing structural constraints and behav-
iSketch Design

... in a software system. It provides a structural modelling tool based on first-order logic [119]. Alloy is targeted at the creation of lightweight models that can be automatically checked for correctness using a constraint solver, Alloy Analyzer.

Figure 4.4: Specification of Sketch-Based Modelling Language Vocabulary in Alloy
Using alloy to validate the syntax of the sketch-based modelling language entails:

1) defining the vocabulary of the language using Alloy’s Signatures model structure. Figure 4.4 depicts the specification of the language in Alloy. The specification is described as follows:

- Define attributes concept as abstract. An attribute in the context of the sketch-based modelling language can either be an actor name and properties, use case description, success scenarios, preconditions and guarantees, source agent, target agent and interaction message.

- Define object concept as abstract. An Object is an Actor, a Use Case, an Agent, or a Target. This definition is consistent with the concrete syntax of the sketch-based modelling language. An actor object has properties which comprise the name, properties, and relationships such as association and generalization. A use case object has properties containing description, success scenarios, preconditions success guarantees and relationships such as include, extend and association. An Agent object has properties comprising source agent, target agent and the message interaction between the two agents.

- Define model concept as abstract. A model can either be a source, intermediate or target model. A source model comprises actors and use cases, an intermediate model comprise agents and the target model comprise targets.

- Finally, define the vocabulary of the language as containing the source, intermediate and target models.

2) Defining constraints that must always hold in the language using Alloy’s Facts model structure. Figure 4.5 presents the constraints.

The constraints are as follows: a) every actor must belong to a source model, b) an actor must not generalize itself, c) actor names must be unique, d) every use case must belong to a source model, e) use case description, success scenarios, preconditions and success guarantees must be unique, f) a use case cannot include or extend itself, g) an agent must belong to an intermediate model and h) a target must belong to a target model.

3) Creating and running an operation to show consistency of the language for one model instance using Alloy’s Predicates model structure. Figure 4.6, Figure 4.7, and Figure 4.8 show the specification and result of the operation in Alloy.
iSketch Design

fact dlRules {
    all a:Actor | one sm:SourceModel | a in sm.actors
    no a:Actor | a = a.generalization
    no a:Actor | a in a.generalization
    all a1, a2: Actor | a1 != a2 -> a1.name != a2.name
    all sm:ActorName | one a:Actor | an in a.name
    all ap:ActorProperties | one a:Actor | ap in a.properties

    all u:UseCase | one sm:SourceModel | u in sm.useCases
    all u1, u2: UseCase | u1 != u2 -> u1.useCaseDescription != u2.useCaseDescription
    all u1, u2: UseCase | u1 != u2 -> u1.useCaseSuccessScenarios != u2.useCaseSuccessScenarios
    all u1, u2: UseCase | u1 != u2 -> u1.useCasePreConditions != u2.useCasePreConditions
    all u1, u2: UseCase | u1 != u2 -> u1.useCaseGuarantees != u2.useCaseGuarantees
    no u:UseCase | u = (u.include | u.exclude)
    no u:UseCase | u in (u.include | u.exclude)

    all a:Agent | one im:IntermediateModel | a in im.agents

    all t:Target | one tm:TargetModel | t in tm.targets
}

Figure 4.5: Specification of Rules to Validate the Vocabulary of Sketch-Based Modelling Language

pred show () {}

run show for 15 but exactly 1 iSketchDSL,
exactly 1 SourceModel, exactly 1 Actor, exactly 1 ActorName,
exactly 1 UseCase, exactly 1 UseCaseDescription, exactly 1 UseCaseSuccessScenarios,
exactly 1 UseCasePreConditions, exactly 1 UseCaseGuarantees,
exactly 1 IntermediateModel, exactly 1 SourceAgent, exactly 1 TargetAgent, exactly 1 InteractionMessage,
exactly 1 Agent, exactly 1 TargetModel, exactly 1 Target

Figure 4.6: Specification of Operations to Check the Consistency of the Vocabulary of Sketch-Based Modelling Language

Executing "Run show for 15 but exactly 1 iSketchDSL, exactly 1 SourceModel, etc.
Solver-set4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
110 vars. 34 primary vars. 136 clauses. 7ms.
Instance found. Predicate is consistent. 15ms.

Figure 4.7: Result showing the Consistency of the Vocabulary of Sketch-Based Modelling Language
Figure 4.8: A Generated Instance Model of Sketch-Based Modelling Language

Running the predicate operations shows the language definition of the sketch-based modelling language consistent and hence validates the abstract and concrete syntax of the language.

4.3.2 Model Transformation Process Flow and Rules

The purpose of this section is to present the process flow and accompanying set of rules for translating the sketch-based users’ requirements into initial software target models. Figure 4.9 shows the process flow.

Figure 4.9: Process Flow for Sketch-Based Model Transformation

The model transformation process starts with generating the concrete syntax of the sketch-based modelling language. This is described in detail in the previous section. Based on the generated concrete syntax, a source model is generated from the hand-
drawn requirements models of a user. Generation of source model entails: 1) the use of the sketch classification results to extract actors and use cases, 2) iteratively extracting the respective properties of the actors and use cases and 3) updating the concrete syntax with the generated source model. Next, a set of agents that acts as intermediate models are generated following two steps, namely: parsing of the structured natural language text of use cases and generation of agents using AGENT-INTERACT-AGENT schema.

Figure 4.10: The Structure of Stanford Parser Output in XML Schema

### 4.3.2.1 Parsing of Use Cases Structured NL Text

Parsing of the structured NL text used for specifying use cases entails identifying structures and semantics of the NL text which, in turn aids in identifying more agents of the system under design in addition to the actors extracted in the source model. Identification of the structures and semantics of NL texts comprises 3 steps: 1) identifying and assigning parts-of-speech (POS) tags to the words in the NL texts 2) creating type dependencies (TDs) or grammatical relations between words in the NL texts and 3) identifying and assigning syntactic structures to the NL texts using a set of syntactic rules.
The syntactic rules defined in this research are similar to those proposed by Chen [120] relating to identification of structures in NL texts.

Steps 1 and 2 for parsing use cases are achieved through the incorporation of a Stanford Parser which does an annotation of use cases’ NL texts and outputs a document in an XML schema that contains tokens and type dependencies between tokens contained in each sentence of the NL texts. Figure 4.10 shows the structure of the document returned by the Stanford Parser.

```
  ▼ object {1}
    ▼ sentences [1]
      ▼ 0 [5]
        structure :value
        index :value
        parse :value
        ▼ basicDependencies [1]
          ▼ 0 [5]
            dep :value
            governor :value
            governorGloss :value
            dependent :value
            dependentGloss :value
        ▼ tokens [1]
          ▼ 0 [12]
            index :value
            word :value
            originalText :value
            lemma :value
            characterOffsetBegin :value
            characterOffsetEnd :value
            pos :value
            ner :value
            speaker :value
            before :value
            after :value
            compoundName :value
```

Figure 4.11: Transformed Structure of Stanford Parser Output in JSON Schema

Since the technical space of the document returned by the Stanford Parser is an XML technical space and the technical space of the generated concrete syntax for model transformation is a JSON technical space, it was necessary to perform an exogenous
model transformation before further processing. This entails a schema conversion from XML to JSON. Figure 4.11 depicts the transformed document in JSON schema.

The tokens returned in the document contain the POS tags for words in each sentence while dependencies provide binary grammatical relations between two tokens. From the available POS tags supported by the Stanford Parser, four of them are used during the model transformation process. They are:

1. **NN**: denotes singular or mass noun.
2. **NNS**: denotes a plural noun.
3. **NNP**: proper singular noun and
4. **NNPS**: proper plural noun.

The current representation of Stanford type dependencies contains approximately 50 grammatical binary relations. A grammatical relation exists between a governor (also known as a regent or a head) and a dependent. Some examples of these relations are nsubj (nominal subject), dobj (direct object), iobj (indirect object) etc. (See [121] for a complete list of relations).

To achieve Step 3 for parsing NL texts to identify and assign a structure to each sentence contained in the texts, 20 grammatical relations described in Table 4.2 were used.

Table 4.2 Grammatical Relations used in iSketch for Parsing NL Text

<table>
<thead>
<tr>
<th>S/No</th>
<th>Grammatical Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>nsubj</td>
<td>A nominal subject. A noun phrase which identifies the syntactic subject of an active clause</td>
</tr>
<tr>
<td>2.</td>
<td>nsubjpass</td>
<td>A passive nominal subject. A noun phrase which represents the syntactic subject of a passive clause</td>
</tr>
<tr>
<td>3.</td>
<td>nmod</td>
<td>Nominal modifiers of noun. It functions as adjunct and it identifies prepositional complements. It holds the relation between the noun/predicate modified by the prepositional complement and the noun introduced by the preposition.</td>
</tr>
<tr>
<td>4.</td>
<td>xcomp</td>
<td>An open clausal complement of a verb or an adjective. It functions as a predicative or clausal complement without its own subject. The reference of the subject is determined by an argument that is external to the xcomp.</td>
</tr>
<tr>
<td>5.</td>
<td>dobj</td>
<td>The direct object of a verb phrase. It functions as the ( accusative) object of the verb.</td>
</tr>
</tbody>
</table>
Table 4.2 Grammatical Relations used in iSketch for Parsing NL Text (cont.)

<table>
<thead>
<tr>
<th>S/No</th>
<th>Grammatical Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.</td>
<td>partmod</td>
<td>A participial form of a verb heading a phrase. It functions as a modifier to the meaning of a noun phrase or a verb.</td>
</tr>
<tr>
<td>7.</td>
<td>infmod</td>
<td>An infinitive form of a verb heading a phrase. Also functions as a modifier to the meaning of a noun phrase or a verb.</td>
</tr>
<tr>
<td>8.</td>
<td>acl</td>
<td>An infinitive form of a verb heading a phrase. Also functions as a modifier to the meaning of a noun phrase or a verb.</td>
</tr>
<tr>
<td>9.</td>
<td>cop</td>
<td>Copula, it functions to link a subject and a nonverbal predicate.</td>
</tr>
<tr>
<td>10.</td>
<td>mark</td>
<td>Marker, it is the word marking a clause as subordinate to another clause</td>
</tr>
<tr>
<td>11.</td>
<td>aux</td>
<td>Auxiliary, a function word that is associated with a verbal predicate that expresses categories. It functions often as a verb.</td>
</tr>
<tr>
<td>12.</td>
<td>neg</td>
<td>Negation modifier, it is the relation between a negation word and the word it modifies.</td>
</tr>
<tr>
<td>13.</td>
<td>advmod</td>
<td>Adverbal modifier, a non-clausal adverb or adverbal phrase that servers to modify a predicate or a modifier word.</td>
</tr>
<tr>
<td>14.</td>
<td>advcl</td>
<td>Adverbial clause modifier, a clause that modifies a verb or another predicate. The dependent of advcl is the main predicate of the clause.</td>
</tr>
<tr>
<td>15.</td>
<td>iobj</td>
<td>An indirect object, it is a nominal phrase that is a core argument of the verb, but it is not subject or direct object.</td>
</tr>
<tr>
<td>16.</td>
<td>ccomp</td>
<td>A clausal complement, it is a dependent clause with an internal subject which functions like an object of a verb or adjective.</td>
</tr>
<tr>
<td>17.</td>
<td>auxpass</td>
<td>A passive auxiliary, it is a non-main verb of the clause which contains passive information.</td>
</tr>
<tr>
<td>18.</td>
<td>nn</td>
<td>A noun compound modifier. NNs are generally written as a single word that in-volves proper names written separately using a dash. It also functions to mark modifiers of non-inflecting noun phrases that generally express profession, rank, position, or assignment.</td>
</tr>
<tr>
<td>19.</td>
<td>appos</td>
<td>An appositional modifier. It is a noun phrase (NP) immediately to the right of the first NP that serves to modify that NP</td>
</tr>
<tr>
<td>20.</td>
<td>amod</td>
<td>An adjectival modifier. It is an adjectival phrase that serves to modify the meaning of the NP.</td>
</tr>
</tbody>
</table>

Using a combination of the grammatical relations listed in Table 4.2 and Table 4.3 above, 33 syntactic rules were defined for this research. 5 out of the 33 syntactic rules were adopted from aToucan [122] while the rest were derived in this research. The syntactic rules are described in Table 4.3.

Table 4.3 Syntactic Rules for Identification of Structures of NL texts used in iSketch

<table>
<thead>
<tr>
<th>S/No</th>
<th>Grammatical Relations</th>
<th>Description</th>
<th>Adopted/Derived</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>SV</td>
<td>It is assigned to a sentence if it contains only nsubj binary relation.</td>
<td>Adopted</td>
</tr>
<tr>
<td>2.</td>
<td>SVADVERBALADJUNCT</td>
<td>It is assigned to a sentence if it contains nsubj, advmod, amod and nmod not in any order.</td>
<td>Derived</td>
</tr>
</tbody>
</table>
Table 4.3 Syntactic Rules for Identification of Structures of NL texts used in iSketch (cont.)

<table>
<thead>
<tr>
<th>S/No</th>
<th>Grammatical Relations</th>
<th>Description</th>
<th>Adopted/Derived</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.</td>
<td>SVAMODDOPO</td>
<td>It is assigned to a sentence if it contains nsubj, amod, dobj and nmod binary relations between various words.</td>
<td>Derived</td>
</tr>
<tr>
<td>4.</td>
<td>SVAUXDO</td>
<td>It is assigned to a sentence if it contains nsubj, dobj and aux binary relations.</td>
<td>Derived</td>
</tr>
<tr>
<td>5.</td>
<td>SVAUXPASSPO</td>
<td>It is assigned to a sentence if there exist nsubj, nsubjpass, auxpass, aux and nmod binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>6.</td>
<td>SVCLAUSE</td>
<td>It is assigned to a sentence if there exist nsubj, nsubjpass, and mark binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>7.</td>
<td>SVCONDITIONAL</td>
<td>It is assigned to a sentence if there exist nsubj and mark binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>8.</td>
<td>SVCONJCLAUSE</td>
<td>It is assigned to a sentence if there exist nsubj, admod and advcl binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>9.</td>
<td>SVCONJINF</td>
<td>It is assigned to a sentence if there exist nsubj, admod, aux, xcomp and ccomp binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>10.</td>
<td>SVDO</td>
<td>It is assigned to a sentence if there exist nsubj and dobj binary relations between its words.</td>
<td>Adopted</td>
</tr>
<tr>
<td>11.</td>
<td>SVDOADVERBIAL</td>
<td>It is assigned to a sentence if there exist nsubj, dobj and advmod binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>12.</td>
<td>SVDOCLAUSE</td>
<td>It is assigned to a sentence if there exist nsubj, dobj, nsubjpass and mark binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>13.</td>
<td>SVDOCOMP</td>
<td>It is assigned to a sentence if there exist nsubj, dobj, aux, cop, infmod, acl and xcomp binary relations between its words.</td>
<td>Adopted</td>
</tr>
<tr>
<td>14.</td>
<td>SVDOCONJCLAUSE</td>
<td>It is assigned to a sentence if there exist nsubj, dobj, admod and advcl binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>15.</td>
<td>SVDOCONJINF</td>
<td>It is assigned to a sentence if there exist nsubj, dobj, admod, aux and xcomp binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>16.</td>
<td>SVDOINF</td>
<td>It is assigned to a sentence if there exist nsubj, dobj, aux, infmod, acl and xcomp binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>17.</td>
<td>SVDONOTINF</td>
<td>It is assigned to a sentence if there exist nsubj, dobj, neg, aux, mark, infmod, acl and xcomp binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>18.</td>
<td>SVDOPART</td>
<td>It is assigned to a sentence if there exist nsubj, dobj and partmod binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>19.</td>
<td>SVDIPO</td>
<td>It is assigned to a sentence if there exist nsubj, dobj and nmod binary relations between its words.</td>
<td>Derived</td>
</tr>
</tbody>
</table>
Table 4.3 Syntactic Rules for Identification of Structures of NL texts used in iSketch (cont.)

<table>
<thead>
<tr>
<th>S/No</th>
<th>Grammatical Relations</th>
<th>Description</th>
<th>Adopted/Derived</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.</td>
<td>SVDOPOAPPOS</td>
<td>It is assigned to a sentence if there exist nsubj, dobj, nmod and appos binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>21.</td>
<td>SVDOPOCLAUSE</td>
<td>It is assigned to a sentence if there exist nsubj, dobj, nmod, and mark binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>22.</td>
<td>SVGERUND</td>
<td>It is assigned to a sentence if there exist nsubj, dobj and xcomp binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>23.</td>
<td>SVINF</td>
<td>It is assigned to a sentence if there exist nsubj, aux and xcomp binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>24.</td>
<td>SVIODO</td>
<td>It is assigned to a sentence if there exist nsubj, iobj and dobj binary relations between its words.</td>
<td>Adopted</td>
</tr>
<tr>
<td>25.</td>
<td>SVIODOPO</td>
<td>It is assigned to a sentence if there exist nsubj, iobj and nmod binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>26.</td>
<td>SVNOTINF</td>
<td>It is assigned to a sentence if there exist nsubj, xcomp, neg and aux binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>27.</td>
<td>SVNOUNADJ</td>
<td>It is assigned to a sentence if there exist nsubj and xcomp binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>28.</td>
<td>SVPASSPO</td>
<td>It is assigned to a sentence if there exist nsubj, nsbjpass, auxpass, and nmod binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>29.</td>
<td>SVPO</td>
<td>It is assigned to a sentence if it contains nsubj, amod, dobj and nmod binary relations between various words.</td>
<td>Derived</td>
</tr>
<tr>
<td>30.</td>
<td>SVPOCOMP</td>
<td>It is assigned to a sentence if there exist nsubj, nmod and ccomp binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>31.</td>
<td>SVPREDICATIVE</td>
<td>It is assigned to a sentence if there exist nsubj and cop binary relations between its words.</td>
<td>Derived</td>
</tr>
<tr>
<td>32.</td>
<td>SVSVCOMP</td>
<td>It is assigned to a sentence if there exist nsubj, nsbj and ccomp binary relations between its words.</td>
<td>Adopted</td>
</tr>
<tr>
<td>33.</td>
<td>SVSVCOMPDOPO</td>
<td>It is assigned to a sentence if there exist nsubj, dobj and nmod binary relations between its words.</td>
<td>Derived</td>
</tr>
</tbody>
</table>

At the end of parsing the NL texts of uses, each sentence in the NL texts is assigned a structure that will be used for agent generation.

### 4.3.2.2 Generation of Agents

Generation of agents entails: 1) identification of group of words that form a single agent, 2) extraction of initial set of agents, their attributes, actions, and relationships, and 3) construction of agent-oriented models. These steps are discussed in what follows:
Identification of group of words that form a single agent

This process uses the parsed NL texts as input to achieve the following: 1) identification of compound nouns in the type dependencies of the parsed NL texts, 2) extraction and concatenation of group of nouns that represent a single concept, and 3) updating the tokens in the parsed NL texts with a compound name that represents the single concept. An example is given in Figure 4.12.

Use Case NL Text: The System reads the ATM Card Number.
Grammatical Relations in Parsed NL Text: det (system-2, The-1) nsubj(reads-3, system-2) root(ROOT-0, reads-3) det(Number-7, the-4) compound(Number-7, ATM-5) compound(Number-7, Card-6) dobj(reads-3, Number-7).
Identification of Compound Grammatical Relations: The grammatical relation based on the dependency type “compound” identifies, compound (Number-7, ATM-5), compound (Number-7, Card-6) as candidates for forming a single concept.
Formation of Single Concept: The compound relations identified above comprise three noun tokens, which are ATM, Card and Number. Hence, the noun tokens will be concatenated to form a single agent, ATM Card Number.
Update of Tokens with Compound Name Value: The token ATM, Card and Number will be updated with the compound name ATM Card Number. Figure 4.13 shows this example.

Figure 4.12: An Example Showing Identification of Compound Name that form a Single Agent
Figure 4.13: Identification of Group of Words that form a Single Agent

Extraction of initial set of agents, their attributes, actions, and relationships.

This process uses the actors extracted from the source model and the tokens generated in the parsed NL texts to generate an initial set of agents using a set 4 of extraction rules. The extraction rules are as follows:

<table>
<thead>
<tr>
<th>Source:</th>
<th>Actors extracted from source model and tokens extracted from parsed NL text</th>
</tr>
</thead>
</table>
| Rule (Pseudocode): | CREATE a list of Temp Agents  
FOR each actor a1 in Actors  
 IF a1 not in Temp Agents THEN  
 add a1 to Temp Agents  
 END IF  
 END FOR  
 FOR each token t1 in Tokens  
 IF pos tag of t1 is a noun and not in Temp Agents THEN  
 add t1 to Temp Agents  
 END IF  
 END FOR |

Figure 4.14: Extraction Rule 1
**Construction of agent-oriented models.**

This process entails the use five translation rules on the type dependencies and structure of the parsed NL texts to construct agent-oriented models using AGENT-INTERACT-AGENT schema. The generated schema contains source agent, target agent and interaction message between the agents. The following figures depict the translation rules:

---

**Source:** Actors extracted from source model

**Rule (Pseudocode):**

FOR each pair of objects o1 and o2 in Temp Agents

IF o1 starts with o2 THEN

CREATE o2 as an agent object with an empty list of attributes

ADD o1 to the attributes of o2

IF Actors contains o2 THEN

RETRIEVE and ADD more attributes of o2 from Actors

END IF

ADD o2 to the list Agents

END IF

END FOR

Figure 4.15: Extraction Rule 2

**Source:** Type dependencies extracted from parsed NL text

**Rule (Pseudocode):**

IF nsubj(has, A) AND dobj(has, B) THEN

CREATE A as an Agent

ADD B to the list of attributes of A

ADD A to the list of Agents

END IF

Figure 4.16: Extraction Rule 3

**Source:** Type dependencies extracted from parsed NL text

**Rule (Pseudocode):**

IF nmod(A, B) OR amod(A, B) OR advcl(A, B) OR poss(A, B) THEN

CREATE B as an Agent

ADD A to the list of attributes of B

ADD B to the list of Agents

END IF

Figure 4.17: Extraction Rule 4

---

**Structures:** SV, SVPREDICATIVE, SVCONDITIONAL

**Rule (Pseudocode):**

source = find in typeDependencies where dependencyType equals 'nsubj'

sourceAgent = source.compoundName ? source.compoundName : source.dependentGloss

targetAgent = sourceAgent

messageInteraction = source.governorGloss

aom = sourceAgent – targetAgent : messageInteraction

**Structures:** SVDO, SVNOUNADJ

**Rule (Pseudocode):**

source = find in typeDependencies where dependencyType equals 'nsubj'

sourceAgent = source.compoundName ? source.compoundName : source.dependentGloss

IF structure is SVDO THEN
Figure 4.19: Translation Rule 2

Structures: SVIODO, SVAUXDO, SVDOADVERBIAL, SVGERUND, SVDOPART, SVPO-COMP

Rule (Pseudocode):

source = find in typeDependencies where dependencyType equals 'nsubj'
sourceAgent = source.compoundName ? source.compoundName : source.dependentGloss

IF structure is SVIODO THEN
    target = find in typeDependencies where dependencyType equals 'iobj'
    mParam = find in typeDependencies where dependencyType equals 'dobj'
ELSE IF structure is SVAUXDO THEN
    target = find in typeDependencies where dependencyType equals 'dobj'
    mParam = find in typeDependencies where dependencyType equals 'aux'
ELSE IF structure is SVDOADVERBIAL THEN
    target = find in typeDependencies where dependencyType equals 'dobj'
    mParam = find in typeDependencies where dependencyType equals 'advmod'
ELSE IF structure is SVGERUND THEN
    target = find in typeDependencies where dependencyType equals 'dobj'
    mParam = find in typeDependencies where dependencyType equals 'xcomp'
ELSE IF structure is SVDOPART THEN
    target = find in typeDependencies where dependencyType equals 'dobj'
    mParam = find in typeDependencies where dependencyType equals 'partmod'
ELSE
    target = find in typeDependencies where dependencyType equals 'nmod'
    mParam = find in typeDependencies where dependencyType equals 'ccomp'

targetAgent = target.compoundName ? target.compoundName : target.dependentGloss
messageParams = mParam.compoundName ? mParam.compoundName : mParam.dependentGloss
messageInteraction = source.governorGloss + (messageParams)
sourceAgent – targetAgent : messageInteraction

Figure 4.20: Translation Rule 3

Structures: SVPO, SVINF, SVADVERBIALADJUNCT, SVPASSPO, SVAUXPASSPO, SVCONJINF, SVCONJCLAUSE

Rule (Pseudocode):

source = find in typeDependencies where dependencyType equals 'nsubj'
sourceAgent = source.compoundName ? source.compoundName : source.dependentGloss

Structures: SVDONOTINF, SVDOPO, SVDOPOAPPOS, SVDOPOCLAUSE, SVDOINF, SVDOCONJINF, SVIDOPO, SVNOSIN, SVSVCOMP, SVSVCOMPDOPO

Rule (Pseudocode):
source = find in typeDependencies where dependencyType equals 'nsubj'
sourceAgent = source.compoundName ? source.compoundName : source.dependentGloss

Figure 4.21: Translation Rule 4
Figure 4.22: Translation Rule 5
The outputs from the intermediate agent-oriented models are a list of agents and their interaction. The model transformation process concludes by generating target models that can be visualized using UML Class and Sequence diagram modelling notation. The generation of Target models consists of the following:

- Migration of initial generated agents to a list of target models.
- Extraction of source agents, target agents and the message interactions from the generated AOMs.
- Addition of all source agents extracted from the AOMs to the list of target models.
- Iteration through the list of target agents extracted from the AOMs and checking if a target agent is an attribute of any target model. If a target agent is found to be an attribute of a target model, the message interaction appended to the target agent in the AOM is added as the target model’s list of actions. Otherwise, the target agent is added to the list of target models with the message interaction added as an action.
- Generation of association relationship between target models based if they exist as agents in the constructed AOMs.
- Generation of a generalization relationship between target models. This is achieved by checking if any target model has a name attribute that are contained in the name of any target agent in the constructed AOMs. If such exists, a generalization relationship is created between the two target models.
- Generation of a Jison Object [123] from the constructed AOMs to create an instance software model represented by UML Sequence diagram notation.
- Generation of Json Object from the list of targets, attributes, actions and relationship to create another instance software model represented by UML Class diagram notation.
4.3.3 Properties of iSketch Sketch-Based Model Transformation Method

Table 4.4 shows the properties of the sketch-based model transformation method.

Table 4.4 Properties of the Sketch-Based Model Transformation Method

<table>
<thead>
<tr>
<th>Properties</th>
<th>Description</th>
</tr>
</thead>
</table>
| Artifacts                               | • End users requirements models  
• Source model  
• Intermediate model  
• Target model                                                                                                                   |
| Number of source and target models      | • Number of source model: 1  
• Number of target models: 2                                                                                                         |
| Technical Spaces                        | • XML returned by the Stanford Parser  
• JSON used through model transformation                                                                                             |
| Endogenous or exogenous transformations  | • Combination of endogenous and exogenous transformations.  
• Exogenous transformation was done during parsing of use case structured NL texts  
• Endogenous transformation was done between the models of the concrete syntax used for model transformation                  |
| Horizontal or vertical transformations  | • Vertical transformation occurred between end users requirements models and the generated initial software models                        |
| Syntactic or semantical transformations  | • Both syntactic and semantic transformation was employed                                                                               |
Chapter 5  iSketch Implementation

5.1 Introduction

A prototype tool based on the sketch-based recognition and model transformation methods has been developed to validate the methods. This chapter discusses the software tool used in this investigation termed, iSketch in four sections. The first section introduces the user interface (UI) and its associated components, the second section describes the meta-model for the UI and its associated components, the third section describes the sketch model transformer and its associated components, and the fourth section presents the steps in using iSketch.

iSketch consists of 11 component (User Interface, Sketch Canvas, Sketch Annotator, Sketch Repository, Sketch Recognition Engine, Sketch Model Transformer, Sketch Model Parser, Sketch Transformation Engine, Sketch Model Interpreter, Sketch Model Generator and Sketch Model Realizer) that enable users to express requirements model represented by Use Case diagram modelling notation via hand-drawn diagram and subsequently transforms the diagram into initial software models represented by UML Class and Sequence diagram.

iSketch is implemented in JavaScript, Dot Net and Python programming languages. It was developed using Visual Studio and PyCharm Integrated Development Environments and contains over 7,000 lines of code.

5.2 User Interface and Related Components

The user interface (UI) component shown in Figure 5.1 provides web-based user-oriented features that allow end users to sketch requirements models, give meaning to their sketches and manage their sketch repository. This component was designed using a set of technologies that comprises HTML 5, CSS 3, Bootstrap 3, Font Awesome Icons, JavaScript, jQuery and RequireJS. These technologies enable iSketch to render well on
a variety of devices and window or screen sizes. The UI component is supported by four other components: Sketch Canvas, Sketch Annotator, Sketch Repository and Sketch Recognition Engine, described as follows.

Figure 5.1: iSketch User Interface. It shows the tool’s Sketch Canvas (A), Annotator (B), Sketch Repository (C), Model Transformer (D) and Other House-Keeping Tools.

5.2.1 Sketch Canvas

Sketch Canvas provides a multi-platform sketching facility that allows end users to express requirements models in Use Case diagram. To sketch Use Case diagram, users can use either formal (computer-drawn) notations, informal (hand-drawn) notations or a mix of both notations. Depending on the device used, users can use a mouse, a stylus or touch gesture to do sketches. Sketch Canvas allows users to resize, rotate, move around, and delete their sketches. Users can save their current sketches and retrieve old sketches for reuse. The Sketch Canvas was designed using HTML 5 Canvas, CSS 3, JavaScript, and jQuery. These technologies enable iSketch to support inputs such as touch, stylus and mouse for drawing notations and requirements models. They further allow iSketch to support features such as drag-and-drop, screen panning, zoom-in and zoom-out.
5.2.2 Sketch Annotator

Sketch annotation enables users to annotate the Use Case diagram drawn on the Sketch Canvas. Users can: 1) give a name to the Use Case diagram, 2) define properties such as actor’s name and attributes, use case’s name and textual specifications, and 3) specify relationships (e.g., include, exclude, association, generalization) between actors and actors, actors and use cases, and use cases and use cases. The Annotator was designed using HTML 5 Forms, CSS 3, Bootstrap 3 Modals, JavaScript, and jQuery.

5.2.3 Sketch Repository

The Sketch Repository comprises Use Case Symbol and Use Case Diagram repositories. Use Case Symbol repository offers an archive for sketch symbols that are typed Actor and Use Case. The repository provides a set of default symbols and enables end users to create and add their own custom hand-drawn symbols. New custom hand-drawn symbols are processed by the Sketch Recognition Engine for symbol classification. Likewise, Use Case Diagram repository offers an archive for storing use case diagrams. It allows reusing of saved use case diagrams, which entails selecting a use case diagram from the Use Case Diagram repository and loading it on the Sketch Canvas. Users can then use the annotated properties of the reused Use Case diagram as is or redefine these properties. Users can save a reused Use Case diagram with the same name or a different name. Saving a reused Use Case diagram with a different name creates and stores a different version of the Use Case diagram. Whereas, saving a reused use case diagram with the same name updates the Use Case diagram with the modifications made. The Sketch Repository was designed using HTML 5 Forms, CSS 3, JavaScript, jQuery UI Accordion and No SQL Database (LokiJS).

5.2.4 Sketch Recognition Engine

The Sketch Recognition engine does two things, 1) it builds, trains, and saves iSketch CNN recognition model based on the architecture described in Section 5.2. Figure 5.2 shows the model. 2) It provides an API through which an unknown sketch can be recognized or classified. The Prediction of a sketch (see Figure 5.3) is done as follows: a) the prediction API receives and saves a post of sketch image, b) the API performs an online pre-processing that entails cropping, colour inversion and, c) reshaping. d) The API retrieves the trained iSketch recognition model; e) performs a prediction and
returns the prediction result. Sketch recognition is used in this research project mainly to classify new symbols as either type Actor or Use Case. This then allows iSketch to know the properties of the symbol to expect during the model transformation process. The Sketch Recognition Engine was implemented using JavaScript and Python.

```python
class iSketchNet:
    @staticmethod
    def build(num_classes):
        model = Sequential()
        model.add(Conv2D(32, (3, 3), input_shape=(1, 32, 32), activation='relu', padding='same'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.15))

        model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.2))

        model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Flatten())

        model.add(Dense(1024, activation='relu'))
        model.add(Dropout(0.3))

        model.add(Dense(4096, activation='relu'))
        model.add(Dropout(0.35))

        model.add(Dense(num_classes, activation='softmax'))

        # Compile model
        optimizer = Adam(1e-3)
        model.compile(loss="categorical_crossentropy", optimizer=optimizer, metrics=["accuracy"])
        return model
```

Figure 5.2: iSketch CNN Recognition Model
In iSketch, the end user can use symbols, links, and labels as elements on the Sketch Canvas to create Use Case diagrams. Symbols and links are typed Elements, i.e., a symbol can be of type actor or use case, whereas, a link can be of type association, generalization, include or extend. Labels are used to add annotations to symbols. The meta-model to support the UI and related components is shown in Figure 6.4. In the following subsections, some of the key meta-classes in the meta-model are described.
5.3.1 Symbol Class

The Symbol class provides a set of functions that support creation and usage of symbol as elements for expressing UCD models on the Sketch Canvas by end users. These functions are described as follows.

Figure 5.4: UI Meta-model and Associated Components of iSketch
**iSketch Implementation**

*Get Symbols.* iSketch calls this function during the process of rendering its User Interface to an end user. The function retrieves saved default and customized symbols from an integrated lightweight NoSQL database (LokiJS). After retrieval of the saved symbols, the Get Symbols function renders the retrieved symbols in the default and customized sections of the Sketch Repository. The rendering of a symbol comprises: 1) creating an image representation of the symbol using HTML 5 image tag, 2) registering drag and drop, and click event listeners on the created image and 3) adding the created image to the default and customized HTML DOMs (Document Object Model) of the Sketch Repository.

```javascript
function getSymbols() {
  var renderCustomSymbols = function() {
    var customSymbols = observe.customSymbolRepository.getCustomSymbols();
    if (customSymbols.size > 0) {
      customSymbols.forEach(function(symbol, key) {
        createImage(symbol.name, symbol.src, symbol.height);
      });
    }
  }

  setTimeout(renderCustomSymbols, 2000);
}
```

Figure 5.5: Code Snippet of Get Symbol function

*Create Symbol.* This function creates a symbol on the Sketch Canvas that an end user drags from or clicks in the Sketch Repository. iSketch goes through three steps to create this symbol on the Sketch Canvas. First, it draws an image of the symbol on the Sketch Canvas using a default width, height and a dropped location for a drag and drop event or an arbitrary location for a click event. Second, it assigns a stringified JSON Object to the drawn image via a custom data attribute. The properties of the assigned Object comprise a unique id that is obtained from an ID Generator, a default name, a type (Actor or Use Case) and attributes (which can either be a list of Actor’s named attributes or a Use Case attribute depending on the symbol’s type). Finally, iSketch creates and links a textual label to the drawn image. The linking of a label to the symbol that is represented by the drawn image is achieved by assigning the symbol’s unique id to the label. After creating the symbol, the end user can then manipulate the symbol and its label by moving and resizing them.
Figure 5.6: Code Snippet of Get Symbol function

Annotate Symbol. The function, Annotate Symbol is called when an end user desires to rename or provide the details of a selected symbol on the Sketch Canvas. Renaming a selected symbol (Actor or Use Case) starts with an end user providing the modified symbol name (using Annotator’s editor facility shown in Figure 3), clicking on the “rename” button, and completes with iSketch updating the name values of the selected symbol’s assigned JSON Object and linked label. However, providing the details of a selected symbol depends on whether the symbol is an Actor, or a Use Case typed symbol. To provide the details of an Actor typed symbol, the end user clicks on “details” button and iSketch displays a modal to the user. On the displayed modal, the end user can add, modify, delete, and save named attributes. Saving the named attributes prompts iSketch to update the attributes value of the JSON Object assigned to the Actor symbol with the named attributes provided by the end user. Likewise, to provide the details of a Use Case typed symbol, the end user also clicks the “details” button and
iSketch Implementation

iSketch displays a modal on which the user can provide a structured natural language requirement of the Use Case. Users can provide details such as the description, scenarios, preconditions and success guarantee of the Use Case. After saving the provided Use Case details, iSketch updates the attributes value of the JSON Object assigned to the Use Case symbol with the provided details.

Delete Symbol. This function enables the end user to remove any selected symbol from the Sketch Canvas. iSketch achieves the deletion of a selected symbol in three steps: First, it retrieves the unique id of the symbol. Second, it uses the unique id to fetch the label linked to the symbol. Finally, it removes the symbol, its linked label, and any other associated links from the Sketch Canvas.

Draw New Symbol. This function enables the end user to be able to create and add custom hand drawn symbol to their Sketch Repository. The function works by loading and displaying to the end user an extended Sketch Canvas modal on which the end user can hand draw, name, and save a new symbol.

Validate New Symbol. After drawing a new symbol, the end user can save the symbol for reuse in the Sketch Repository. On saving the symbol, iSketch uses the Validate New Symbol function to validate the symbol before adding it to the Sketch Repository. The validation of a new symbol comprises a syntactic and semantic symbol validation.

The syntactic symbol validation process ensures that the new symbol’s name has not been used to annotate any existing symbol in the Sketch Repository. iSketch achieves this type of validation by using the Check If Symbol Exists function. The function filters through the entire symbol names that are existing in the Sketch Repository and confirms if the name of the newly drawn symbol exists or not. If the name exists, iSketch notifies the user and requests for another name. Entering another name takes the symbol through a repeated syntactic symbol validation process.

Once the symbol passes the syntactic validation phase, iSketch further takes the symbol through a semantic validation process. The aim of the semantic symbol validation is to find the type of the new symbol. To achieve semantic symbol validation, iSketch uses the Recognize Symbol function of the Recognition Classifier class. The Recognize symbol function works by taking as an input, an image of the new symbol and then sends
the image for type prediction. The prediction is handled by the Sketch Recognition Engine component of iSketch. It achieves prediction on an input image through 1) an online data pre-processing that comprises cropping and colour inversion, 2) loading of the saved and trained iSketch CNN model and 3) using iSketch CNN model to predict the type of the new symbol. The prediction result is then returned as a Recognition Result. The Recognition Result contains the type of the matched symbol and the corresponding matching score.

If the newly hand-drawn symbol passes both the syntactic and semantic validation, iSketch then saves the symbol into the Sketch Repository of the user.

Save New Symbol. This carries out two tasks during the process of the saving the hand-drawn symbol of the end user. These tasks are as follows: 1) the function creates an image representation of the symbol, registers drag and drop, click event listeners to the created image, and adds the image to the HTML DOM of the customized section of the Sketch Repository. 2) It persists the symbol to the NoSQL database of iSketch. Doing this makes the new symbol to be available for retrieval during any subsequent reload of the iSketch tool.

5.3.2 Link Class

The Link class provides a function, Draw Link, which enables the end user to create links between symbols. The end user can create links between two different use cases, a use case and an actor or between two different actors. The process of creating a link between symbols is as follows: 1) the end user selects the “From Symbol”, which can be an actor or a use case. 2) The end user selects a relationship type from Annotator’s editor facility. A relationship type can be an association, generalization, include or exclude and it determines the type of link that will be drawn between two selected symbols. 3) The end user selects the “To Symbol”, which can also be an actor or a use case. On selecting the “To Symbol”, iSketch will draw a link that corresponds to the relationship type chosen by the user between the “From Symbol” and “To Symbol”. Further, iSketch will create and assign a JSON Object to the drawn link via a custom data attribute. The properties of this assigned object comprise the respective ids of the “From and To Symbols”, and the link type.
5.3.3 UCD Model Class

The UCD Model class provides a set of functions that enable the end user to manage, transform or download their UCD model. These functions are described as follows:

*Save UCD Model.* This function enables the end user to save their work done at any stage during the drawing and annotation of a UCD model. The process of saving a UCD model is carried out in two phases. First phase, the end user 1) provides the name of the UCD model using the Annotator’s editor facility and 2) clicks on the “save use case diagram” button on iSketch’s tool kit. Second phase, iSketch uses the Save UCD Model function to carry out the following: 1) Retrieval of the respective JSON Object of all the symbols and links used as elements in expressing the UCD model and creating a Symbol Link Combo model. 2) Creating a UCD model that comprises the name of the Use Case diagram (provided by the end user) and the Symbol Link Combo model (created by iSketch). 3) Persisting the created UCD model to iSketch database and 4) Creating and adding an HTML5 radio input tag with the name of the UCD model to the HTML DOM of the UCD Sketch Repository. Once the UCD model is saved, the end user can subsequently update additional work done to the model.

*Update UCD Model.* iSketch uses this function to save end user’s modifications of an existing UCD model. To save any modifications made, iSketch uses the Update UCD Model function to 1) Create a Symbol Link Combo model that represents the modified UCD model. 2) Retrieve the persisted UCD model using the name of the modified UCD model. 3) Replace the Symbol Link Combo model of the retrieved UCD model with the Symbol Link Combo model created in (1), and 4) persist the change made to its database.

*Delete UCD Model.* iSketch uses the Delete UCD Model function to enable the end user to delete any saved UCD model. To delete a UCD model, the end user starts by clicking a “delete” button in the UCD Sketch Repository. The Delete UCD Model function prompts the user to confirm the deletion request. If the user confirms the continuation of the deletion request, iSketch, retrieves the persisted UCD model using the name of the UCD model, deletes the retrieved UCD Model from its database and then remove the HTML5 radio input tag that corresponds to the deleted UCD model from the HTML DOM of the UCD Sketch Repository.
iSketch Implementation

Load UCD Model. This function enables an end user to load any saved UCD model from the UCD Sketch Repository onto the Sketch Canvas for reuse, modifications, or transformation. The process of loading any UCD Model is follows: 1) the end user selects the HTML radio input tag that has the name of the UCD Model to be loaded from the UCD Sketch Repository, 2) iSketch gets the name of the selected HTML radio input tag and retrieve the corresponding UCD Model from its database. 3) iSketch uses the Symbol Link Combo model of the retrieved UCD Model, the Create Symbol function of the Symbol Class and the Draw Link function of the Link class to iteratively draw symbol and link elements on the Sketch Canvas. The process of drawing symbols and links automatically by iSketch on the Sketch Canvas follows the aforementioned processes in Sections 6.3.1 and 6.3.2 above but with iSketch taking the place of the all end user’s inputs.

Download Document. Through this function, iSketch enables end users to download an initial Natural Language (NL) requirements document that contains the use cases of their UCD model. The process of downloading a NL requirements document entails the following: 1) the end user clicks the “initial requirements document” button on iSketch’s toolkit. 2) iSketch iterates through all the symbol elements of end user’s UCD model and filters out symbols that of use case type. 3) iSketch iterates through the respective associations (links) of all the filtered use case symbols and retrieves their respective primary actors and the use cases they include and or extend. 4) iSketch creates a JSON object for each use case with properties comprising the name, description, success scenarios, preconditions, success guarantees, primary actors, and include and extend use cases if any. 5) iSketch downloads the created JSON objects as a pdf file for the end user.

Transform UCD Model. This function enables the end user to transform their drawn and annotated UCD model to a set of initial software models. The Transform UCD Model function is called when the end user clicks on the “model transformer” button on iSketch’s toolkit to start the transformation process.
5.4 Sketch Model Transformer and Related Components

The sketch model transformer and related components are responsible for transforming user hand-drawn requirements into initial software models. They implement the sketch-based model transformation method discussed in Section 5.3. They are discussed as follows:

5.4.1 Sketch Model Transformer

The Sketch Model Transformer uses the Transform UCD Model function to 1) create a concrete syntax based on the sketch-based modelling language that will be used for the sketch to model transformation, 2) update the source model in the concrete syntax with extracted actors and use cases from the Sketch Canvas and 3) invoke back-end oriented-features to generate the intermediate and target models in the concrete syntax. This component was designed using HTML 5 and JavaScript and is supported by five components: Sketch Model Parser, Sketch Transformation Engine, Sketch Model Interpreter, Software Model Generator and Software Model Realizer. These back-end features are described as follows:

5.4.2 Sketch Model Parser

The Sketch Model Parser starts a rule-based object-oriented analysis of the source model in the concrete syntax as follows: 1) it extracts all the actors and use cases in the source model, 2) it iterates through the extracted use cases to identify and concatenates their respective success scenarios sentences and 3) it stores all the extracted actors and concatenated success scenarios sentences in a JSON Object with attributes Actors and Scenario Sentences. The Sketch Model Parser was designed using JavaScript and JavaScript Object Notation (JSON). These technologies enable the Parser to transmit the Scenario Sentences to the Use Case Interpreter for further processing in form of data object. Figure 6.7 shows an example of the input and output of the Sketch Model Parser.
5.4.3 Sketch Transformation Engine

The Sketch Transformation Engine acts as the core for the sketch to software model transformation. It implements 3 set of rules that aid the transformation process. The rules are: 1) syntactic rules for assigning structures to each scenario sentence provided by the Sketch Model Parser, 2) extraction rules for generating initial set of agents in the Intermediate Model and 3) translation rules for generating the final set of agents and their interaction using an AGENT-INTERACT-AGENT schema. Rules provided by the Sketch Transformation Engine are used by the Sketch Model Interpreter and Software Model Generator components of iSketch to carry out their functions. Section 5.3.2 discussed the syntactic, extraction and translation rules in detail.

5.4.4 Sketch Model Interpreter

The Sketch Model Interpreter incorporates Stanford Parser and uses the syntactic rules provided by the Sketch Transformation Engine to process each sentence in the Scenario Sentences. The interpreter performs three tasks: 1) identifying and assigning Parts-of-Speech (POS) tag to each word in a sentence (e.g., noun, verb, adverb, etc.), 2) creating Type-Dependencies (TDs) or grammatical relations between words in respective sentences, and 3) identifying and assigning a structure to each sentence in the Scenario Sentences. The interpreter uses the Stanford Parser to achieve tasks 1 and 2 while it uses the syntactic rules provided by the Sketch Transformation Engine to achieve task 3.
The following is an example of sentence structure identification and assignment:

- **Use Case Scenario Sentence [End-User]**: The System reads the ATM Card Number.

- **Grammatical Relations [Stanford Parser]**: det (system-2, The-1) nsubj(reads-3, system-2) root(ROOT-0, reads-3) det(Number-7, the-4) compound(Number-7, ATM-5) compound(Number-7, Card-6) dobj(reads-3, Number-7)

- **Identification and Assignment of Sentence Structure [Sketch Model Parser]**: The grammatical relations contain one nominal subject (nsubj) and one direct object (dobj). Based on Syntactic Rule #10, the Sketch Model Interpreter will assign a structure SVDO to the use case scenario sentence.

After assigning structures to the Scenario Sentences, the interpreter provides a structured specification, represented in JSON format as an output. The interpreter was designed using technologies comprising JSON, Stanford Parser, XML, C# and .Net Web API.

### 5.4.5 Software Model Generator

The Software Model Generator uses the output of the Sketch Model Interpreter to generate the Intermediate Model in the concrete syntax used for model transformation by performing the following tasks:

*Identify group of words that form a single concept.* The generator iterates through the type dependencies list of each sentence, identifies compound dependency type and use it to extract, concatenate group of nouns that represent a single concept, and update the individual noun token with the formed single concept as compound name. An example is given in Section 5.3.2

*Extract Initial Set of Entity Classes, their attributes, actions and relationships.* After the identification and formation of single concepts, the generator iterates through the set of Actors extracted by the Sketch Model Parser and scenario sentence tokens produced by the Sketch Model Interpreter to extract all nouns and store them in a list called Entity Objects. Subsequently, the generator uses the set of Extraction rules provided by the Sketch Transformation Engine on the Entity Objects, Actors as well as the type dependencies of scenario sentences to obtain a set of Agents and their attributes.
Construct Agent Oriented Models (AOMs). The generator performs a final iteration on the type dependencies and the corresponding structure of the Scenario Sentences produced by the Sketch Model Interpreter and applies the set of Translation rules provided by the Sketch Transformation Engine to construct Agent Oriented Models (AOMs) using an AGENT-INTERACT-AGENT schema. The schema of the AOMs consists of a source agent, target agent, a message interaction between the two agents and a message parameter. The Software Model Generator was designed using JavaScript, Jison and JSON.

5.4.6 Software Model Realizer

The Software Model Realizer performs the last step in the model transformation process discussed in Section 5.3.2 to generate the Target Model in the concrete syntax used for the transformation. It uses two sets of data to generate two types of targets. The sets of data used are the set of Agents and the AOMs generated by the Software Model Generator. It uses these agent data to generate software target models represented by UML Class and Sequence Diagrams. The Software Model Realizer component was designed using HTML 5, CSS 3, Bootstrap 3, JavaScript, JSON and Scalable Vector Graphics (SVG).

5.4.7 Other Features

In addition to the above described components, iSketch also has some standard “housekeeping” facilities:

- Clear Screen enables users to clear the Use Case diagram drawn on the Sketch Canvas.
- Screen Panning allows users to view off-screen space of the Sketch Canvas.
- Zoom-in allows users to examine their Use Case diagram more closely.
- Zoom-out allows users to examine essential parts of their Use Case diagram, rather than the details.
- Delete allows users to remove from the Sketch Canvas any sketch they want to.
- Save allows users to save their Use Case diagram for a later reuse.
- Initiate Requirements Document allows users to generate a textual document that contains the use cases specified in their Use Case diagram.
5.5 Steps in Using iSketch

The typical steps in using iSketch consist of the following steps:

**Step 1:** An end user browses iSketch on the web to capture or express the requirements of a system under design. iSketch opens and displays to the user, a user interface showing a Sketch Canvas, a Sketch Annotator panel, Sketch Repositories, and a set of Tools as shown in Figure 6.1. Using the displayed user interface, the end user draws the Use Case diagram requirements model describing the set of actions (use cases) the system under design should perform in collaboration with one or more external users (actors) of the system. To start drawing the Use Case diagram, the user opens the Default Symbol Repository, selects, and drags and drops a use case symbol on the Sketch Canvas to start specifying the actions that system under design should perform (Figure 5.8).

![Figure 5.8: A section of iSketch. It shows a dragged and dropped use case symbol](image)

The user makes use of the editor facility of the Sketch Annotator to change the name of the use case by typing the preferred name and clicking the rename button (Figure 5.9). After giving a name to the use case, the user adds an actor that will be responsible for the use case. To add an actor, the user can either use the default actor symbol provided by iSketch or create a custom symbol. To create a custom symbol, the user opens the Customized Symbols repository of iSketch and clicks the New Symbol button. iSketch displays a Sketch Canvas for the User to hand draw their symbol. After sketching the new symbol, the user annotates the name of the symbol and clicks Save. Symbol button (Figure 6.10).
iSketch Implementation

Figure 5.9: A section of iSketch. It shows the change of name from use case to “withdraw funds”.

Figure 5.10 A section of iSketch. It shows a Sketch Canvas for creating custom symbols.

On clicking the save button, iSketch validates the syntax and semantics of the symbol drawn to ensure that a similar symbol does not exist. If the symbol is valid, iSketch saves the user’s symbol to the Customized Symbols repository (Figure 5.11). If the symbol is not valid, iSketch prompts the user to rename the symbol if a symbol with the same name already exists. With the custom symbol now available, the user selects and drags and drops the symbol on the Sketch Canvas (Figure 5.12). Next, the user creates an association between the actor (customer) and use case (withdraw funds) by selecting Association from the Relationship Type drop down on the Annotator’s editor and clicking on the withdraw funds use case symbol (Figure 5.13).
Figure 5.11: A section of iSketch. It shows the custom symbol “customer” newly created by the user.

Figure 5.12: A section of iSketch. It shows the actor “customer” added to the Sketch Canvas.

Figure 5.13: A section of iSketch. It shows the created association relationship between the actor (customer) and use case (withdraw funds).
The end user adds more actors and use cases of the system under design by going through a similar process explained so far. After creating the requirements model, the user iteratively elicits the details of the actors and use cases of the system under design. To add details to the actor (customer), the user selects the customer symbol on the Sketch Canvas and clicks on details button on the Sketch Annotator’s editor. On the modal dialog that follows, the user adds and saves the attributes of the customer (Figure 5.14). Likewise, to add details to the use case (withdraw funds), the user selects the withdraw funds symbol on the Sketch Canvas and clicks the details button on the Annotator’s editor. The user supplies and saves the description, success scenarios, preconditions and success guarantee of the withdraw funds use case using natural language text (Figure 5.15).

Figure 5.14: A section of iSketch. It shows added attributes of the actor (customer).

After annotating the actors and use cases of the system under design with their respective details, the user saves the Use Case diagram requirements model by typing the name of the system on the Sketch Annotator and clicking save use case diagram button on iSketch’s tool panel. On clicking the save button, iSketch saves the user’s diagram into the Use Case repository for a later reuse (Figure 5.16).

**Step 2:** After creating the Use Case diagram requirements model, the end user initiates a sketch to initial software model transformation process by clicking the Model Transformer button (marked D in Figure 5.1).

**Step 3:** iSketch uses its Sketch Model Transformer and the related components (described in Section 6.4) to generate and display to the user, the initial software models
represented by UML Sequence and Class diagrams shown in Figure 5.17 and Figure 5.18, respectively.

Figure 5.15: A section of iSketch. It shows adding of natural language text details to the use case (withdraw funds).

Figure 5.16: A section of iSketch. It shows the saved use case diagram (bank atm system).
Figure 5.17: It shows a Sequence diagram software model generated from the Use Case diagram in Figure 5.16.
Figure 5.18: It shows a Class diagram conceptual model generated from the Use Case diagram in Figure 5.16
Chapter 6  Evaluation

6.1 Introduction

The aim of this chapter is to provide answers to the research question RQ4 “How well can deep learning and AOMs support sketch recognition and model transformation?” To answer this question, the aim of this chapter is decomposed into the following objectives:

1. **To evaluate iSketch Sketch-Based Recognition Method via the use of Stroke Labelling (SL).** The choice of stroke labelling as the selected metric to evaluate the sketch recognition performance of iSketch Sketch-Based Recognition Method stems from the outcome of the review of 20 sketch recognition approaches done in Chapter 2. The review revealed that 10 out of 20 of the approaches were evaluated using stroke labelling metric. This will help to provide answer to RQ4. Further to evaluating

2. **To evaluate iSketch Sketch-Based Model Transformation Method via the use of Precision and Recall.** Transformation approaches are evaluated based on the qualities of their outputs. However, evaluating the transformation of requirements models is a challenging task. This is because there is no standard evaluation method in existence. Reason being that models obtained from the transformation of requirements models are not good or bad, rather, they are useful or less useful [124]. The thesis of this project is that it is feasible to automatically transform hand-drawn requirements models into initial software models using the Sketch-Based Model Transformation method. Hence, to measure the classification accuracy of the UML class and sequence diagram elements created through the implementation of the method, this research project opted for precision and recall metrics. This will help to provide answer to RQ4.

3. **To compare iSketch with Related Work.** This compares iSketch with related sketch recognition approaches, model transformation approaches and sketching
software tools that enable end users to express requirements models using hand-drawn diagrams. This will help to provide answer to RQ5.

![Sketch Images](image.png)

Figure 6.1: Some examples of sketch images in iSketch dataset used to evaluate iSketch Sketch-Based Recognition method

### 6.2 Evaluation of iSketch Sketch-Based Recognition Method

This section presents the evaluation of iSketch CNN that implements the proposed Sketch-Based Recognition method for sketch classification and recognition task. First, a description of the datasets that are used to evaluate the proposed iSketch CNN is given. Second, the iSketch CNN training detail used for the evaluation is described and finally, results of the evaluation are presented.

#### 6.2.1 Datasets

**iSketch Dataset.** The scope of this research is within the early phase of RE. Precisely, it concerns expressing requirements models during early RE using UML Use Case hand-drawn symbols or shapes. This therefore necessitated having a sketch dataset that comprises the common symbols of a Use Case diagram. In the absence of the required dataset in literature or web, three postgraduate students were contacted to assist with sketching of Use Case diagram symbols using iSketch. 480 sketch images hand-drawn...
with a mix of uni-strokes and multi-strokes were obtained and categorised into 6 classes namely: actor, use case, decision, database, oval, and process. The 480 sketch images were further increased to 6,000 images using data augmentation. Figure 6.1 shows one example of the sketch images from each of the six classes.

6.2.2 Training Detail

Using iSketch dataset, two sets of experiments were conducted to evaluate the sketch classification and recognition performance of the proposed iSketch CNN with and without colour inversion. In the experiments, the sketch datasets were augmented using the sketch pre-processing pipeline discussed in Chapter 4. Each augmented dataset was then split into three parts for training, testing and validation, containing 80%, 10% and 10% of sketches, respectively. The training dataset which comprises 80% of the dataset is used by the network for learning and fitting of the network. The validation dataset of 10% provided an unbiased evaluation of the network fit on the training dataset while tuning its hyper parameters. The test dataset constitutes the remaining 10% and it was used to evaluate the final network fit on the training dataset. The split ratio between the training, validation and test dataset ensures a balanced variance between the hyper parameter estimates and the classification performance statistics of the network.

In all the training, the following were used: categorical cross entropy on Tensor Flow was used as a loss function for single label classification or stroke labelling, a learning rate of 0.0001, a batch size of 128 and 10 epochs. Each training took a maximum of 50 minutes approximately on Google Collaborative platform that provides a free 12GB NVIDIA Tesla K80 GPU for academic projects.

6.2.3 Results

This section presents the sketch classification and recognition performance of iSketch CNN on iSketch dataset using stroke labelling accuracy as metric. The metric compares the class label assigned to a given stroke against a ground truth that exists in a database.

\[
\text{Stroke Labelling Accuracy} = \frac{\text{Assigned Stroke Class Label}}{\text{Correct Stroke Class Label}}
\]
Evaluation

The sketch classification and recognition performance results are presented in what follows:

1. Sketch Recognition Performance on iSketch Dataset without Colour Inversion

After 10 epochs, iSketch CNN reached sketch recognition accuracy of 91.46% on iSketch dataset without the use of colour inversion. Figure 6.2 shows the plot of the training loss vs validation loss as well as the training accuracy vs validation accuracy.

Figure 6.2: iSketch Sketch Recognition Result on iSketch Dataset without Colour Inversion

An evaluation of the network with the test dataset puts iSketch sketch classification and recognition accuracy at 89.91%.
2. Sketch Recognition Performance on iSketch Dataset without Colour Inversion

After 10 epochs, iSketch CNN reached sketch recognition accuracy of 98.75% on iSketch dataset with the use of colour inversion. Figure 6.3 shows the plot of the training loss vs validation loss as well as the training accuracy vs validation accuracy.

Figure 6.3: iSketch Sketch Recognition Result on iSketch Dataset without Colour Inversion

An evaluation of the network with the test dataset puts iSketch sketch classification and recognition accuracy at 97.29%.

6.2.4 Discussion

The plots in Figure 6.2 and Figure 6.3 shows a good network fit in the sense that: 1) training and validation losses were both low, have minimum differences and decreased
Evaluation

as the training progressed, 2) training and validation accuracies increased as the training progressed, and 3) the loses were decreasing as the accuracies were increasing.

Without the use of the proposed colour inversion in the sketch pre-processing pipeline, iSketch achieved a recognition accuracy of 89.91% on 600 test datasets while with colour inversion, the recognition accuracy was increased to 97.29% on the same test datasets (an improvement of 7.38% in recognition accuracy). Comparing both accuracies, with and without colour inversion, one can see the important role colour inversion played when using deep learning on sketch-based images. Also, given that iSketch dataset was made up of symbols drawn with a combination of single and multiple strokes, the high recognition rate achieved suggests that iSketch Sketch-Based Recognition Method is stroke independent.

6.3 Evaluation of iSketch Sketch-Based Model Transformation Method

6.3.1 Method

To evaluate the performance of iSketch Sketch-Based Model Transformation method, 10 different use case scenarios (see Appendix A.1) were used. These use case scenarios were taken from three different software systems, NextGen [125] (a point of sale system), Elevator [126] (an elevator system) and ATM [127] (a banking system). The corresponding software models of the use case scenarios were obtained and considered to be created by domain experts (e.g., renowned textbook authors). However, a single gold standard model for any given use case scenario does not exist, as different domain experts will usually produce different software models. These models cannot be categorized as strictly correct or incorrect, in spite of this, they are usually categorized as useful or less useful [124]. This research project assumed that the software models from the domain experts are useful and hence they served as benchmarks in the evaluation process.

The initial software models (Appendix A.2) generated by iSketch were compared with the software models generated by the domain experts (Appendix A.3) to get a confusion matrix. A confusion matrix is a table that is often used to describe the performance of a classification model on a test data (the software models generated by
iSketch) for which the true values are known (from each software model generated by the domain experts). Each of the model elements contained in the software models generated by iSketch was used as a classification. These model elements comprise source agents, target agents, and message interaction between agents in a Sequence diagram model, and classes, methods, associations, and generalizations in a Class diagram models. The confusion matrix distinguishes between the predicted values in the initial software models generated by iSketch and the real values in the software models generated by domain experts. The metrics used for computing the confusion matrix are as follows:

**True Positive**: The model elements that are included in both the software models generated by iSketch and domain experts.

**False Positive**: The model elements that are included in the software models generated by iSketch but not included those generated by the domain experts.

**False Negative**: The model elements that are not included in the software models generated by iSketch but are included in those generated by the domain experts.

From the values in the confusion matrix, reports that include three perform metrics (Recall, Precision and F-score) from the software models generated by iSketch and the domain experts were created.

Recall is the number of model elements retrieved (True Positive) divided by the number of model elements generated by the domain experts (True Positive + False Negative):

\[
Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}
\]

Precision is the number of model elements correctly retrieved (True Positive) divided by the number of model elements generated by iSketch (True Positive + False Positive):

\[
Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}
\]

F-score, a measure of the accuracy of iSketch, is the harmonic average of the Precision and Recall:
Evaluation

\[ F - score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \]

A Recall value of 0% for a particular classification means that there are no model elements obtained from the software models generated by the domain experts that matched the software models generated by iSketch. Whereas a Recall value of 100% means that all the model elements obtained from the software models generated by the domain experts are present in the software models generated by iSketch for a particular classification.

A Precision value of 0% for a particular classification means that no model elements obtained from the software models generated by iSketch matched those of the software models generated by the domain experts. Whereas a Precision value of 100% means that all the model elements obtained from the software models of iSketch are present in the software models generated by the domain experts.

A Recall value and a Precision value of 100% for a particular classification of iSketch model elements means that both iSketch and the domain experts generated the same model.

6.3.2 Results

This section presents the performance measurement in terms of precision and recall values for model elements of the software models (Sequence and Class diagram) generated by iSketch when compared to those generated by domain experts.

6.3.2.1 Sequence Diagram Performance Results

Figure 6.4, Figure 6.5, and Figure 6.6 show three charts with the results of the recall and precision values of iSketch’s Sequence diagram model elements after comparison with the Sequence diagram software models generated by the domain experts. Each chart represents a classification of iSketch’s Sequence diagram model elements (Source Agent, Target Agent, and Interaction Messages). Each point in the charts represents recall values on the y-axis and precision values on the x-axis for each software model.

Figure 6.4 related to the Source Agents in iSketch’s Sequence diagram models shows that after comparison with the software models generated by the domain experts, most of the Recall values are higher than the Precision values (7 of 10). The ranges of values for Source Agents are [85.71%, 100%] for Recall and [66.67%, 100%] for Preci-
Likewise, Figure 6.5 related to the Target Agents in iSketch’s Sequence diagram models shows that after comparison with the models generated by the domain experts, most of the Recall values are higher than the Precision values (9 of 10). The ranges of values for Target Agents are [80.00%, 100%] for Recall and [78.57%, 100%] for Precision. However, Figure 6.6 related to the Interaction Messages generated in iSketch’s Sequence diagram models, shows that the number of Recall values that are higher than the Precision values are 5 out of 10. The ranges of values for Messages are [75.00%, 100%] for Recall and [81.82%, 100%] for Precision.

Table 6.1 shows the mean values of Recall, Precision and F-Score of the graphs of each classification of iSketch’s Sequence diagram model. The Target Agents generated by iSketch have the best result in recall (96.45%) and precision (89.74%). In Recall, the next best result is obtained by Source Agents (96.07%) followed by Interaction Messages (90.85%). Whereas, in Precision, the next best result after Target Agents is Interaction Messages (88.26%) followed by Source Agents (87.17%)

Figure 6.4: Recall and Precision values of Source Agents Generated by iSketch
Figure 6.5: Recall and Precision values of Target Agents Generated by iSketch

Figure 6.6: Recall and Precision values of Message Interaction between Agents Generated by iSketch
Table 6.1: Mean Values for Recall, Precision and F-score of iSketch’s Sequence Diagram

<table>
<thead>
<tr>
<th>Classification</th>
<th>Recall (%)</th>
<th>Precision (%)</th>
<th>F-Score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Agents</td>
<td>96.07</td>
<td>87.27</td>
<td>90.45</td>
</tr>
<tr>
<td>Target Agents</td>
<td>96.45</td>
<td>89.74</td>
<td>92.77</td>
</tr>
<tr>
<td>Interaction Messages</td>
<td>90.85</td>
<td>88.26</td>
<td>90.45</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>94.46</strong></td>
<td><strong>88.42</strong></td>
<td><strong>91.22</strong></td>
</tr>
</tbody>
</table>

6.3.2.2 Class Diagram Performance Results

Figure 6.7, Figure 6.8, and Figure 6.9 show three charts with the results of the recall and precision values of iSketch’s Class diagram model elements after comparison with the Class diagram software models generated by the domain experts. Each chart represents a classification of iSketch’s Class diagram model elements (Classes, Methods, Associations and Generalizations). Each point in the charts represents recall values on the y-axis and precision values on the x-axis for each analysis model. Figure 6.7 related to the Classes in iSketch’s Class diagram models shows that after comparison with the software models generated by the domain experts, most of the Recall values are higher than the Precision values (8 of 10). The ranges of values for Classes are [28.57%, 100%] for Recall and [33.33%, 83.33%] for Precision. In like manner, Figure 6.8 related to the Methods in iSketch’s Class diagram models shows that after comparison with the models generated by the domain experts, most of the Recall values are higher than the Precision values (7 of 10). The ranges of values for Methods are [36.36%, 100%] for Recall and [40.00%, 88.46%] for Precision. However, Figure 6.9 related to the Associations and Generalizations generated in iSketch’s Class diagram models, shows that the number of Recall values that are higher than the Precision values are 5 out of 10. The ranges of values for Associations and Generalizations are [0.00%, 100%] for Recall and [0.00%, 80.00%] for Precision.

Table 6.2 shows the mean values of Recall, Precision and F-Score of the graphs of each classification of iSketch’s Class diagram model. The Methods generated by iSketch have the best result in recall (80.43%) and precision (71.18%). In Recall, the next best result is obtained by Classes (77.52%) followed by Associations and Generali-
Evaluations (49.21%). In addition, in Precision, the next best result after Methods is Classes (62.94%) followed by Associations and Generalizations (37.53%).

Figure 6.7: Recall and Precision values of Classes Generated by iSketch

Figure 6.8: Recall and Precision values of Classes’ Methods Generated by iSketch
Evaluation

Figure 6.9: Recall and Precision values of Classes’ Association and Generation Relationships Generated by iSketch

Table 6.2 Mean Values for Recall, Precision and F-score of iSketch’s Class Diagram

<table>
<thead>
<tr>
<th>Classification</th>
<th>Recall (%)</th>
<th>Precision (%)</th>
<th>F-Score (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classes</td>
<td>77.52</td>
<td>62.94</td>
<td>67.63</td>
</tr>
<tr>
<td>Methods</td>
<td>80.43</td>
<td>71.18</td>
<td>73.17</td>
</tr>
<tr>
<td>Association and Gener-</td>
<td>49.21</td>
<td>37.53</td>
<td>41.85</td>
</tr>
<tr>
<td>alizations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>69.05</strong></td>
<td><strong>57.21</strong></td>
<td><strong>60.88</strong></td>
</tr>
</tbody>
</table>

6.3.3 Discussion

Ten use case scenarios, each extracted from three systems, were used for the evaluation of iSketch Sketch-Based Model Transformation method. The use case scenarios range between 60 and 298 words, and between 6 and 29 sentences in size. The average word size is 129 words, and the average sentence size is 13 sentences.
The recall values of 100% showed that iSketch generated all the model elements that were also created in the software models of the domain experts. However, the recall values of 0% showed that the Associations and Generalizations model elements created in the software models of the domain experts did not match those generated by iSketch. This occurred in 2 out of the 10 use case scenarios used for evaluating iSketch. Furthermore, recall values that are lower than 100% indicated that iSketch only generated some of the model elements created in the software model of the domain experts.

A further analysis of these results indicated that in cases where iSketch had Recall values that are lower than 100%, the domain experts may have created the corresponding software models using unstated or implicit information about the domain. This was noticed in the creation of Source Agents in 3/10 models, Target Agents in 3/10 models, and Interaction Messages in 6/10 models for Sequence diagram, and Classes in 5/10 models, Methods in 8/10 models, and Associations and Generalization in 6/10 models for Class diagram.

**6.4 Comparison of iSketch with Related Work**

This section compares iSketch tool with related sketch recognition approaches, model transformation approaches and sketching software tools reviewed in the literature.

**6.4.1 Comparison with Related Sketch Recognition Approaches**

This section compares the performance of iSketch sketch recognition method with that of 20 sketch recognition approaches reviewed in Section 2.2 for processing hand-drawn diagrams or sketches for recognition. Table 6.3 shows the comparison. The comparison shows the following:

- iSketch is the only sketch-based approach for processing and recognizing hand-drawn use case diagram symbols.
- iSketch and Casella et [28] are the only recognition approaches that take use case diagram symbols as input for sketch recognition.
- iSketch achieved the highest recognition accuracy of 97.29% when compared with the recognition approaches that used stroke labelling as a metric for evaluation sketch recognition performance.
### Table 6.3 Comparison with Related Sketch Recognition Approaches

<table>
<thead>
<tr>
<th>Recognition Approach</th>
<th># of Approaches</th>
<th>Approach Name</th>
<th>Evaluation Measure</th>
<th>Findings/Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sketch-based</td>
<td>1</td>
<td>iSketch</td>
<td>Stroke labelling</td>
<td>Achieved a recognition accuracy of 97.29% with colour inversion and 89.91% without colour inversion in recognising use case diagram elements</td>
</tr>
<tr>
<td>Gesture-based</td>
<td>1</td>
<td>SkApp [19]</td>
<td>Stroke labelling</td>
<td>Achieved recognition rate of 92.00% for UML Class symbols and 88.50% for graphical user interface elements</td>
</tr>
<tr>
<td>Stroke-based</td>
<td>13</td>
<td>Bresler et al. [24]</td>
<td>Stroke labelling, symbol segmentation and classification</td>
<td>Achieved 96.70% average stroke labelling and 92.83% average symbol segmentation and classification recognition rate for flowchart and finite automata</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Costagliola et al. [25]</td>
<td>None</td>
<td>No performance results</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deufemia et al. [26]</td>
<td>Precision and recall</td>
<td>Achieved 84.83% average precision rate and 88.50% average recall rate for UML class diagram symbols</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Hammond et al. [27]</td>
<td>Stroke labelling</td>
<td>Achieved 96.00% weighted recognition rate on 15 primitive shapes of UML class diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUMLOW [9]</td>
<td>Stroke labelling</td>
<td>Achieved 84.30% recognition rate on 14 primitive shapes of UML class diagram</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Casella et al. [28]</td>
<td>Stroke labelling</td>
<td>Achieved 89.87% in classifying UML use case diagram elements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Brieler et al. [34]</td>
<td>Recognition processing time</td>
<td>Achieved 360ms in processing and recognising 39 components of a Petri net diagram.</td>
</tr>
</tbody>
</table>
|                      |                 | Yuan and Jin [36]   | Usability test                      | • 74.40% pretty good, 20.60% good and 5.00% disappointing Ease of Use  
• 84.40% pretty good, 14.40% good and 1.20% disappointing Efficiency  
• 93.80% pretty good, 6.20% good and 0.00% disappointing Natural Interaction  
• 57.50% pretty good, 26.30% good and 16.20% disappointing Reliability  
• 78.10% pretty good, 18.80% good and 3.10% disappointing Users’ Satisfaction  |
|                      |                 | Tahuti [8]          | Usability test                      | Ease of use of 3.5 and ease of editing of 4.2 out of a scale of 5 |
|                      |                 | Hse and Newton [37] | Stroke labelling                   | Achieved average recognition rate of 96.90%, 91.98% and 93.55% with Support Vector Machines, Minimum Mean Distance and Nearest Neighbour classifiers respectively |
|                      |                 | Sezgin and Davis [40] | Stroke labelling and sketch processing time | Achieved 96.50% recognition rate for a set of 10 objects |
|                      |                 | SketchGIS [41]      | None                               | No performance results |
|                      |                 | Lank et al. [42]    | Annotation test                     | No performance results |
Table 6.3 Comparison with Related Sketch Recognition Approaches (cont.)

<table>
<thead>
<tr>
<th>Recognition Approach</th>
<th># of Approaches</th>
<th>Approach Name</th>
<th>Evaluation Measure</th>
<th>Findings/Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Hammond and Davis [43]</td>
<td>Stress test</td>
<td>Achieved a total time of 15660 milliseconds to recognize 56 shapes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PaleoSketch [46]</td>
<td>Stroke labelling</td>
<td>Achieved a recognition rate of 98.56% in classifying eight primitive shapes: arc, circle, curve, ellipse, helix, line, polyline, and spiral shapes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Yu and Cai [47]</td>
<td>Usability test</td>
<td>Achieved an average of 96.00% recognition rate for primitive shapes, polylines, and arcs.</td>
</tr>
<tr>
<td>Image-based</td>
<td>1</td>
<td>Arueleba et al. [51]</td>
<td>None</td>
<td>No performance results.</td>
</tr>
</tbody>
</table>

### 6.4.2 Comparison with Related Model Transformation Approaches

This section compares the input, output, and evaluation measure of iSketch model transformation method with the model transformation approaches reviewed in Section 2.3. The following summarises the comparison:

- iSketch is the only model transformation approach that achieved an automatic transformation into not only metamodel but initial UML class and sequence diagram software model. This means that the outputs of iSketch model transformation method do not require an upload or import into another system for the generation of UML class and sequence diagram software models as currently required by the reviewed related model transformation approaches.
- iSketch is the only model transformation approach that evaluated the performance of the transformation using precision and recall.
- In contrast with FlexiSketch [55], [56] that carried out a user study for the evaluation of the achieved model transformation, iSketch was not evaluated using user study. This is largely due to the time constraint posed by the PhD duration.
6.4.3 Comparison with Related Sketching Software Tools

The comparison with related sketching software tools is based on the required features that enable sketching of requirements models and transformation into initial software models.

6.4.3.1 Sketching

iSketch, like the related sketching tools enable users to express requirements models on a sketch canvas using either hand-drawn or computer-drawn use case symbols. Similar to three tools (FlexiSketch [55], [62], [63], OctoUML [3], [64] and SketchiXML [68]), iSketch’s user interface supports the use of various platforms ranging from mobile phone, tablets, laptops to desktop PCs. Furthermore, like most of the tools, iSketch provides a Sketch Repository feature that enables users to save their sketches for a later reuse. However, unlike three of the related tools (FlexiSketch [55], [62], [63], SUMLOW [9] and OctoUML [3], [64]) that support co-located collaborative sketch, the current implementation of iSketch only supports sketching by a single user.

To support users during the sketching process, iSketch like most of the tools provides Sketch Annotation and Recognition features. However, the rationale and implementation of these features diverge from related tools. First, the Sketch Annotation feature allows annotation of not just the name of the symbols used in expressing a require-
ments model, but also a list properties and natural language description of the requirements represented by a symbol. The rationale behind this feature is it to enable an incremental requirements elicitation process with a tool from just sketching of rough ideas to a fine-tuned and detailed requirements model that represent the system under design. The resulting requirements model can then be saved, transformed, or downloaded as a requirements document from iSketch. Second, unlike the related tools that make use of Sketch Recognition to auto-complete, suggest or beautify hand-drawn diagrams of end users, iSketch applied sketch recognition as a starting point for sketch to model transformation. With sketch recognition, iSketch can know what type of sketch was drawn by the user and hence what sort of model transformation process will be required. Furthermore, iSketch unlike the other tools does not depend on stroke information of a sketch. It disregards the stroke type (single or multi-strokes), order of aspect ratio of a sketch.

6.4.3.2 Model Transformation Process

Only one of the related software sketching tools has attempted model transformation. The tool, FlexiSketch [55], [56] uses a lightweight meta-modelling approach that supports the definition of types for symbols, links and cardinalities for link types. The tool uses this definition to semi-automatically generate a meta-model from hand-drawn requirements models. The process of generating a meta-model by the tool follows: 1) sketching of requirements models on a sketch canvas by an end user, 2) assigning of types to sketched models via annotation by the end user. The annotations provide the basic structure of the meta-model generated by FlexiSketch. 3) Cardinalities between sketched models are inferred to complete the meta-model. The generated meta-model can then be exported into a CASE tool for further processing. However, this introduces a manual process of recreating software models from scratch which is time consuming and error prone. In addition, the information provided by the meta-model is mainly basic structures do not contain more detailed information about the system under design. For example, the rationale behind having an actor in a system or what the actor’s role in the system would be may be forgotten by the time the meta-model is generated or manually transformed into an initial software model.

iSketch employs a sketch-based modelling language and an intermediate model transformation process to address the shortcoming of the current model transformation process. The sketch-based modelling language provided the abstract and concrete syn-
tax for sketch-based model transformation and how such a language can be validated for completeness using a declarative formal specification language. Through the concrete syntax of the model transformation method, an intermediate transformation takes place from the user’s requirements model to an intermediate model and finally to target models. At the intermediate model transformation step, iSketch outputs a set of Agents and Agent-Oriented Models (AOMs) that represent the knowledge captured from the user’s requirements model. AOMs provide the interactions between agents. The advantage of this is reusability. The outputted Agents and their respective interactions can then be mapped to target models of choice. In the case of this research, the Agents and AOMs were mapped into generating UML Class and Sequence diagrams.

Table 6.5 gives a summary of features of iSketch when compared to existing related sketching software tools.

<table>
<thead>
<tr>
<th>Category</th>
<th>Tool</th>
<th>Hand Sketch</th>
<th>Computer-Aided Sketch</th>
<th>Multi-Platform Sketch</th>
<th>Collaborative Sketch</th>
<th>Sketch Annotation</th>
<th>Sketch Recognition</th>
<th>Sketch Repository</th>
<th>Sketch Dependency Type</th>
<th>Model Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Early RE Tools</td>
<td>iSketch</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Any</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>FlexiSketch [55], [62]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single Stroke</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Calico [5], [7]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Not Applicable</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SUMLOW [9]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Multi-Stroke</td>
<td></td>
</tr>
<tr>
<td>General Modeling Tool</td>
<td>OctoUML [3], [64]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single Stroke</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Donaldson et al. [65]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Single Stroke</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Lank et al. [42], [66]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Multi-Stroke</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tahuti [8]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Single Stroke</td>
<td></td>
</tr>
</tbody>
</table>
## Evaluation

Table 6.5 A Summary of Features Supported by Related Sketch Tools (cont.)

<table>
<thead>
<tr>
<th>Category</th>
<th>Tool</th>
<th>Hand Sketch</th>
<th>Computer-Aided Sketch</th>
<th>Multi-Platform Sketch</th>
<th>Collaborative Sketch</th>
<th>Sketch Annotation</th>
<th>Sketch Recognition</th>
<th>Sketch Repository</th>
<th>Sketch Stroke Type</th>
<th>Model Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface Tools</td>
<td>FreeForm [67]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single Stroke</td>
</tr>
<tr>
<td></td>
<td>SketchiXML [68]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
<tr>
<td></td>
<td>SILK [1], [69]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Single Stroke</td>
</tr>
<tr>
<td>Sketching Tools</td>
<td>MaramaSketch [70]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
<tr>
<td></td>
<td>Brieler et al. [34]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
<tr>
<td></td>
<td>SketchREAD [72]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
<tr>
<td></td>
<td>GDE [73]</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>Multi-Stroke</td>
</tr>
</tbody>
</table>
Chapter 7 Conclusions

7.1 Research Achievements

This research has made the following contributions:

1. A systematic study of 40 existing sketch recognition and model transformation techniques and tools.

2. A sketch-based recognition method. A method that uses deep learning specifically, Convolutional Neural Network (CNN) for sketch classification and recognition. The recognition achieved with the method does not constrain the user to using a type of sketch stroke (single or multiple), stroke order or aspect ratio when drawing use case diagram symbols. The method uses: a) colour inversion in an offline data augmentation pipeline that pre-processes training data and 2) a CNN network designed for sketch or hand-drawn diagram for sketch classification and recognition. This contribution led to the following:

   a) An off-the shelf library for either online or offline sketch recognition.
   b) An annotated sketch dataset of hand-drawn UML shapes that can be used to train a system for Use Case diagram symbol sketch recognition.

3. A sketch-based model transformation method. A method for the automatic construction of initial software models from hand-drawn requirements models to support the early stages of requirements elicitation and modelling. The approach uses Agent-Oriented Models (AOMs) as an intermediate model between a source model (a user’s drawing) and target models (auto-generated initial software models). This contribution led to the following:

   a) Semantics and syntactic validation of the sketch-based modelling language using Alloy, a declarative formal specification language.
   b) A set of extraction and translation rules for constructing AOMs from user’s sketch diagram and structured use case texts.
c) Taxonomy of sketch transformation methods for generating meta-models from hand-drawn requirements models.

4. The development of a prototype software tool (iSketch). A prototype tool developed to validate the feasibility of the sketch-based recognition and model transformation methods. It was developed using a combination of JavaScript, Dot Net and Python programming languages. It enables users to express requirements using hand drawings and supports them to subsequently transform the diagram into initial software models.

7.2 Open Challenges

The following challenges have been identified with regards to processing hand-drawn diagrams for sketch recognition and model transformation:

- Integrating sketch-based recognition and model transformation methods. One major gap that this research has attempted to fill is how sketch recognition can be used to drive a sketch to model transformation. The initial challenge here was finding a related work that in some way addresses this challenge. The review of related work revealed that some mainly addressed challenges with sketch recognition while others barely scratch the surface of model transformation via the generation of meta-models based on the symbols and links used by users during requirements modelling. Defining a sketch-based modelling language helped to address this challenge. The abstract syntax of the language defined a Source Model based on types (actors or use cases). Each type having a set of expected properties that will then be extracted during the model transformation process. Sketch recognition played a major role in classifying user’s sketch into class actor or use case. This is subsequently used at the beginning of the model transformation process where the properties of actors and use cases are extracted and processed for the generation of initial software models. This approach can be adapted to support many (source) to many (target) modelling notations.

- Availability of UML Use Case dataset for the evaluation iSketch Sketch-Based recognition method. With the research scope been within the early phase of RE, there was an initial challenge on what dataset to make use of to evaluate the proposed sketch recognition method. The TU-Berlin sketch benchmark dataset pro-
vided objects that are not related to the objects used in modelling notations. But, there was a need to compare the performance of the proposed method with other methods that used TU-Berlin as benchmark. However, the challenge of getting a more focused dataset was still there. A search through literature yielded nothing. Finally, the challenge was addressed by providing a feature in iSketch through which UML Use Case hand-drawn symbols were gathered from three post graduate students.

- **Evaluation of iSketch Sketch-Based model transformation method.** Transformation approaches are evaluated based on the quality of their output models. The initial software models that formed the output of the sketch-based model transformation method are not good or bad; rather, they are relevant or less relevant. Consequently, an appropriate method that evaluates the relevance of the output models of the proposed model transformation method should be adapted. In this research, a stand measure of relevance was adapted from information retrieval based on precision and recall metrics.

### 7.3 Limitations and Future Work

The results achieved through evaluation of the proposed method justified aim and objectives of this research. However, the limitations of this research need to be made explicit thereby opening avenues for future work. The limitations are summarised as follows:

- **Limitations to use structured scenario sentences.** The current version of iSketch mainly supports the use of structured scenario sentences for specifying in detail the requirements of the system under design. It requires the user to using a subject-object-predicate structure when writing scenario sentences to get good model transformation results.

- **Limitation of the extraction and translation rules for model transformation.** The rules that the model transformation process of iSketch depends on, may not contain all necessary rules for the generation of AOMs, which are required for the construction of sequence diagram and class diagram initial software models. Despite this limitation, it was noticed that the rules performed better when structured scenario sentences were used.
Conclusions

- **Limitations in collaborative sketching.** iSketch currently does not support collaborative modelling of requirements as some of the existing related sketching tools (FlexiSketch, SUMLOW and Knight) do. This is mainly because; the aim of this research is to demonstrate the feasibility of the proposed methods in a single-user device. However, collaborative modelling of requirements will allow participants in either a co-located or geographically distributed elicitation meetings to have concurrent editing access to a synchronized Sketch Canvas.

- **Limitations of auto-generated software models.** The initial software models produced by iSketch are mostly first cut sequence and class diagram models, which may need to be refined by requirements analysts.

- **Limitation in the notation for modelling requirements.** The current version of iSketch limits users to mainly the use of use case diagram for expressing requirements. It also limits user to hand-draw the symbols of the use case diagram and then subsequently drag and drop the symbols on to the canvas where the use case diagram is drawn. Hence, end users need to use drag and drop and know the syntax of use case diagram to benefit from the features of iSketch.

Possible future endeavours from this research can be summarised into the following:

- A support for sketching and recognition of hand-drawn requirements models without the need to drag and drop the symbols needed for the diagram.
- A support for more notations that can be used for expressing requirements models as well as generating initial software models. This will not restrict users to understanding a particular type of notation before they can benefit from iSketch.
- A refactoring of the rules used in the model transformation process to support any type of NL structure for specifying scenario sentences.
References

References


References


[34] F. Brieler, and M. Minas, "Recognition and processing of hand-drawn diagrams using syntactic and semantic analysis." pp. 181-188.


References


References


References


References


References


References


Appendix A

A.1. Use Case Scenarios used to evaluate iSketch Sketch-Based Model Transformation Method

NextGen Point of Sale (POS) System

Use Case #1: Process Cash Sale

Use Case Success Scenarios:

1. Cashier makes a new sale on the system.
2. Cashier enters item identifier on the system.
3. The System records sale line item.
4. System presents item description to the Cashier.
5. System presents price to the Cashier.
6. System presents the running total to the Cashier.
7. Cashier makes payment to the System.
8. The System saves logs of completed sale.
9. System presents receipt to Cashier.

Use Case #2: Process Credit Sale

Use Case Success Scenarios:

1. Cashier makes a new sale on the system.
2. Cashier enters item identifier on the system.
3. The System records sale line item.
4. System presents item description to the Cashier.
5. System presents price to the Cashier.
6. System presents the running total to the Cashier.
7. System gets taxes of the sale from a Tax Calculator.
8. System presents the total including taxes to the Cashier.
9. Cashier makes credit payment to the System.
10. The system asks for payment approval from an external Credit Authorization Service System.
11. The system receives payment approval from the Credit Authorization Service System.
12. The system signals the Cashier.
13. The System records the credit payment including the payment approval

**Elevator – An Elevator System**

**Use Case #3: Select Destination**

**Use Case Success Scenarios:**

1. The Elevator Button Request arrives at the Elevator Button Interface.
2. The Elevator Button Interface sends the Elevator Request to the Elevator Manager.
3. The Elevator Manager sends update to Elevator Status Plan.
4. The Elevator Status Plan sends acknowledgement to the Elevator Manager.
5. The Elevator Manager sends an Elevator Commitment message to the Scheduler.
6. The Elevator Manager sends a direction message to the Elevator Control.

**Use Case #4: Request Elevator**

**Use Case Success Scenarios:**

1. The Floor Button Request arrives at the Floor Button Interface.
2. The Floor Button Interface sends a Service Request to the Scheduler.
3. The Scheduler sends a Scheduler Request to the Elevator Manager.
4. The Elevator Manager sends Update message to the Elevator Status Plan.
5. The Elevator Status Plan sends acknowledgement to the Elevator Manager.
6. The Elevator Manager sends an Elevator Commitment message to the Scheduler.
7. The Elevator Manager sends a direction message to the Elevator Control

**Use Case #5: Stop Elevator at Floor**
Appendix A

Use Case Success Scenarios:

1. The Arrival Sensor Interface receives an input from the arrival sensor external entity.
2. The Arrival Sensor Interface sends floor number to the Elevator Control.
3. The Elevator Control sends a Check This Floor message to the Elevator Status Plan.
4. The Elevator Status Plan sends Approaching Requested Floor message to the Elevator Control.
5. The Elevator Control sends stop commands to the Motor Interface.
7. The Direction Lamp Interface switches on the direction lamp.
8. The Motor Interface sends Stop Motor Command to the motor.
10. Motor Interface sends Elevator Stopped message to the Elevator Control.
11. The Elevator Control sends Open Door command to the Door Interface.
12. The Elevator Control sends Off Elevator Lamp message to the Elevator Lamp Interface.
13. The Elevator Lamp Interface sends an Elevator Lamp Output to the external lamp.
14. The Elevator Control sends Arrived message to the Elevator Status Plan.
15. The Elevator Control sends Arrived message to the Scheduler.
16. The Door Interface sends Open Door Command to the door.
17. The Door Interface receives the Door Response.
18. The Door Interface sends a Door Opened message to the Elevator Control.
19. The Elevator Control starts a timer.

Use Case #6: Dispatch Elevator

Use Case Success Scenarios:

1. Elevator Status Plan sends Up Request message to the Elevator Control.
2. Elevator Control sends a Close Door command to Door Interface.
3. Elevator Control sends Off Up Floor Lamp to the Floor Lamp Interface.
4. The Door Interface sends a Close Door Command to the door.
5. The door sends a Closed Door Response to the Door Interface.
6. The Door Interface sends a Door Closed message to Elevator Control.
7. Elevator Control sends Up Command to the Motor Interface.
8. Elevator Control sends Off Up Direction Lamp request to the Direction Lamp Interface.
9. Direction Lamp Interface switches of the direction lamp.
10. The Motor Interface sends a Start Up Motor Command to the motor.
12. The Motor Interface sends an Elevator Started message to Elevator Control.
13. Elevator Control sends a Departed message to the Elevator Status Plan.
14. Elevator Control sends a Departed message to the Scheduler.

**ATM - A Banking System**

**Use Case #7: ATM Client Validate Pin**

**Use Case Success Scenarios:**

1. The Card Reader Interface sends the card input data to the ATM Card.
2. Card Reader Interface sends the Card Inserted to ATM Control.
3. Control sends the Get PIN to Customer Interaction.
4. Customer Interaction displays the PIN Prompt message to the ATM Customer.
5. ATM Customer inputs the PIN number to the Customer Interaction.
6. Customer Interaction asks for card data from ATM Card.
7. ATM Card provides the card data to the Customer Interaction.
8. Customer Interaction sends Card Id, PIN, Start Date, and Expiration Date, to the ATM Transaction.
9. ATM Transaction sends the PIN Validation Transaction to Customer Interaction.
10. Customer Interaction sends the PIN Entered message to ATM Control.
11. ATM control sends a Validate PIN request to the Banking Service.
12. Banking Service validates the PIN.
14. ATM Control sends the Display Menu message to Customer Interaction.
15. ATM Control sends Update Status message to the ATM Transaction.
16. Customer Interaction displays a menu that shows the Withdraw, Query and Transfer options to the ATM Customer.
Appendix A

Use Case #8: Banking Service Validate PIN

Use Case Success Scenarios:

1. ATM Client sends the Validate PIN request to the PIN Validation Transaction Manager.
2. PIN Validation Transaction Manager sends a Validate Card Id, PIN message to the Debit Card.
3. Debit Card validates the customer entered PIN.
4. Debit Card sends Valid PIN response to PIN Validation Transaction Manager.
5. PIN Validation Transaction Manager asks for account numbers from Card Account.
6. Card Account sends valid account numbers to PIN Validation Transaction Manager.
7. PIN Validation Transaction Manager saves the transaction with the Transaction Log.
8. PIN Validation Transaction Manager sends a Valid PIN response to the ATM Client.

Use Case #9: ATM Client Withdraw Funds

Use Case Success Scenarios:

1. ATM Customer enters Withdrawal selection to Customer Interaction.
2. ATM Customer enters account number to Customer Interaction.
3. ATM Customer enters withdrawal amount to Customer Interaction.
4. Customer Interaction sends the customer selection to ATM Transaction.
5. ATM Transaction responds with the Withdrawal Transaction details to Customer Interaction.
6. Customer Interaction sends the Withdrawal Transaction request to ATM Control.
7. ATM Control sends a Withdrawal transaction request to the Banking Service.
8. ATM Control sends a Display Wait message to Customer Interaction.
10. Banking Service sends a Withdrawal Approved response to ATM Control.
11. ATM Control sends a Dispense Cash message to Cash Dispenser Interface.


12. ATM Control sends an Update Status message to ATM Transaction.
13. Cash Dispenser Interface sends the Withdraw Amount to ATM Cash.
14. ATM Cash sends a positive Cash Response to the Cash Dispenser Interface.
15. Cash Dispenser Interface sends the Dispenser Output command to the Cash Dispenser.
16. Cash Dispenser Interface sends the Cash Dispensed message to ATM Control.
17. ATM Control sends print message to Receipt Printer Interface.
18. ATM Control sends the Display Cash Dispensed message to Customer Interaction.
20. ATM Control sends a Confirm Cash Dispensed message to the Banking Service.
21. Receipt Printer Interface asks transaction data from ATM Transaction.
22. ATM Transaction sends the transaction data to the Receipt Printer Interface.
23. Receipt Printer Interface sends the Printer Output to the Receipt Printer.
24. Receipt Printer Interface sends the Receipt Printed message to ATM Control.
25. ATM Control sends the Eject message to Card Reader Interface.
27. Card Reader Interface sends the Card Ejected message to ATM Control.
28. ATM Control sends the Display Ejected message to the Customer Interaction.
29. Customer Interaction displays the Card Ejected prompt to the ATM Customer.

**Use Case #10: Banking Service Withdraw Funds**

**Use Case Success Scenarios:**

1. ATM Client sends Withdrawal request to the Withdrawal Transaction Manager.
2. Withdrawal Transaction Manager sends Check Daily Limit message to Debit Card.
3. Debit Card validates the daily limit for cash withdrawal.
5. Withdrawal Transaction Manager sends a message to Account.
6. Account responds with Withdrawal Status to Withdrawal Transaction Manager.
7. The Withdrawal Transaction Manager sends Update Daily Total to Debit Card.
8. Withdrawal Transaction Manager saves the transaction with the Transaction Log.
9. Withdrawal Transaction Manager responds with Withdrawal Status to the ATM Client.

A.2. iSketch Models Generated for each Use Case Scenario used to Evaluate iSketch Sketch-Based Model Transformation Method

NextGen Point of Sale (POS) System

Use Case #1: Process Cash Sale

![Sequence Diagram Generated by iSketch for Process Cash Sale (NextGen Point of Sale System)]
Use Case #2: Process Credit Sale

Figure 0.2: Class Diagram Generated by iSketch for Process Cash Sale (NextGen Point of Sale System)

Figure 0.3: Sequence Diagram Generated by iSketch for Process Credit Sale (NextGen Point of Sale System)
Figure 0.4: Class Diagram Generated by iSketch for Process Credit Sale (NextGen Point of Sale System)
Appendix A

Elevator – An Elevator System

Use Case #3: Select Destination

Figure 0.5: Sequence Diagram Generated by iSketch for Select Destination (Elevator System)
Figure 0.6: Class Diagram Generated by iSketch for Select Destination (Elevator System)

Use Case #4: Request Elevator
Figure 0.7: Sequence Diagram Generated by iSketch for Request Elevator (Elevator System)
Figure 0.8: Class Diagram Generated by iSketch for Request Elevator (Elevator System)
Use Case #5: Stop Elevator at Floor

Figure 0.9: Sequence Diagram Generated by iSketch for Stop Elevator at Floor (Elevator System)
Figure 0.10: Class Diagram Generated by iSketch for Stop Elevator at Floor (Elevator System)
Appendix A

Use Case #6: Dispatch Elevator

Figure 0.11: Sequence Diagram Generated by iSketch for Dispatch Elevator (Elevator System)
Figure 0.12: Class Diagram Generated by iSketch for Dispatch Elevator (Elevator System)
Appendix A

ATM - A Banking System

Use Case #7: ATM Client Validate Pin

Figure 0.13: Sequence Diagram Generated by iSketch for ATM Client Validate Pin (Banking System)
Figure 0.14: Class Diagram Generated by iSketch for ATM Client Validate Pin (Banking System)
Use Case #8: Banking Service Validate PIN

Figure 0.15: Sequence Diagram Generated by iSketch for Banking Service Validate Pin (Banking System)
Figure 0.16: Class Diagram Generated by iSketch for Banking Service Validate Pin (Banking System)
Use Case #9: ATM Client Withdraw Funds

Figure 0.17: Sequence Diagram Generated by iSketch for ATM Client Withdraw Funds (Banking System)
Figure 0.18: Class Diagram Generated by iSketch for ATM Client Withdraw Funds (Banking System)
Appendix A

Use Case #10: Banking Service Withdraw Funds

Figure 0.19: Sequence Diagram Generated by iSketch for Banking Service Withdraw Funds (Banking System)
Figure 0.20: Class Diagram Generated by iSketch for Banking Service Withdraw Funds (Banking System)
A.3. Benchmark Models used to Evaluate iSketch
Sketch-Based Model Transformation Method

NextGen Point of Sale (POS) System

Use Case #1: Process Cash Sale

**Figure 10.2. SSD for a Process Sale scenario.**

![Sequence Diagram](view_full_size_image)

Figure 0.21: Benchmark Sequence Diagram for Process Cash Sale (NextGen Point of Sale System)

Use Case #2: Process Credit Sale
Figure 0.22: Benchmark Sequence Diagram for Process Credit Sale (NextGen Point of Sale System)
Figure 0.23: Benchmark Consolidated Class Diagram for NextGen Point of Sale System
Appendix A

Elevator – An Elevator System

Use Case #3: Select Destination

![Sequence Diagram for Select Destination](image1)

Figure 0.24: Benchmark Sequence Diagram for Select Destination (Elevator System)

Use Case #4: Request Elevator

![Sequence Diagram for Request Elevator](image2)

Figure 0.25: Benchmark Sequence Diagram for Request Elevator (Elevator System)
Appendix A

Use Case #5: Stop Elevator at Floor

Figure 0.26: Benchmark Sequence Diagram for Stop Elevator at Floor (Elevator System)

Use Case #6: Dispatch Elevator

Figure 0.27: Benchmark Sequence Diagram for Dispatch Elevator (Elevator System)
Appendix A

Figure 0.28: Benchmark Consolidated Class Diagram for Elevator System
Appendix A

ATM - A Banking System

Use Case #7: ATM Client Validate Pin

Figure 0.29: Benchmark Sequence Diagram for ATM Client Validate Pin (Banking System)
Use Case #8: Banking Service Validate PIN

Figure 0.30: Benchmark Sequence Diagram for Banking Validate Pin (Banking System)
Appendix A

Use Case #9: ATM Client Withdraw Funds

Figure 0.31: Benchmark Sequence Diagram for ATM Client Withdraw Funds (Banking System)
Appendix A

Use Case #10: Banking Service Withdraw Funds

Figure 0.32: Benchmark Sequence Diagram for Banking Service Withdraw Funds
(Banking System)
Figure 0.33: Benchmark Consolidated Class Diagram for Banking System
Figure 0.34: Benchmark Consolidated Class Diagram Attributes for Banking System