Semi-automated assessment of programming languages for novice programmers

This item was submitted to Loughborough University's Institutional Repository by the/author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfilment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: [https://dspace.lboro.ac.uk/2134/33669](https://dspace.lboro.ac.uk/2134/33669)

Publisher: © Selim Buyrukoglu

Rights: This work is made available according to the conditions of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0) licence. Full details of this licence are available at: [https://creativecommons.org/licenses/by-nc-nd/4.0/](https://creativecommons.org/licenses/by-nc-nd/4.0/)

Please cite the published version.
Semi-Automated Assessment of Programming Languages for Novice Programmers

By

Selim Buyrukoglu, BSc MSc

A Doctoral Thesis
Submitted in partial fulfilment of the requirements for the award of

Doctor of Philosophy Degree of Loughborough University

July 2018
Abstract

There has recently been an increased emphasis on the importance of learning programming languages, not only in higher education but also in secondary schools. Students of a variety of departments such as physics, mathematics and engineering have also started learning programming languages as part of their academic courses. Assessment of students’ programming solutions is therefore important for developing their programming skills. Many Computer Based Assessment (CBA) systems utilise multiple-choice questions (MCQ) to evaluate students’ performance. However, MCQs lack the ability to comprehensively assess students’ knowledge. Thus, other forms of programming solutions are required to assess students’ knowledge. This research aims to develop a semi-automated assessment framework for novice programmers, utilising a computer to support the marking process. The research also focuses on ensuring the consistency of feedback. A novel marking process model is developed based on the semi-automated assessment approach which supports a new way of marking, termed ‘segmented marking’. A study is carried out to investigate and demonstrate the feasibility of the segmented marking technique. In addition, the new marking process model is developed based on the results of the feasibility study, and two novel marking process models are presented based on segmented marking, namely the full-marking and partial-marking process models. The Case-Based Reasoning (CBR) cycle is adopted in the marking process models in order to ensure the consistency of feedback. User interfaces of the prototype marking tools (full and partial) are designed and developed based on the marking process models and the user interface design requirements. The experimental results show that the full and partial marking techniques are feasible for use in formative assessment. Furthermore, the results also highlight that the tools are capable of providing consistent and personalised feedback and that they considerably reduce markers’ workload.

Keywords: Computer-Based Assessment, Novice Programmers, Formative Assessment, Segmented Marking, Case-Based Reasoning, Programming Solutions, Full and Partial Marking.
To my father
Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university.

Selim Buyrukoglu
July 2018
Acknowledgements

I would like to express my sincere gratitude to my supervisors Dr. Fırat Batmaz and Dr. Russell Lock for the continuous support they provided during my PhD study and related research, and for their patience, motivation, and immense knowledge. Their guidance helped me during the entire research process and writing of this thesis. I could not have imagined having better supervisors and mentors for my PhD study.

I wish to thank Prof. Ray Dawson for graciously accepting to be the reviewer for the annual reviewer, and for his feedback which helped me clarify and improve a number of aspects in my research. I am also grateful to fellow research students for their useful discussions, debates and the warm company.

I would like to thank the technical staff members Mr. Gurbinder Singh Samra and Mr. Kip Sahnsi for their kind assistance and prompt responses to my numerous requests for technical support. My sincere thanks also goes to the secretarial staff members, especially Mrs Judith Poulton, for their cordiality and assistance with regards to departmental matters. I would also like to extend my sincere thanks to all of my friends in Loughborough.

Many thanks are also due to all people that took part in the testing of the systems developed during the course of the research. Special thanks to Dr. Roger Stone for his help in the testing of the tools.

I would like to express my sincere gratitude to the Republic of Turkey’s Ministry of National Education for the funding provided for my research studentship and the warm support provided throughout.

Very special thanks go to my wife Gonca and daughter Feyza for their love, support, understanding and the many sacrifices they made throughout the period of my study. Also, I offer my grateful thanks to my family and siblings for their ever-constant love and support.
Table of Contents

List of Tables .......................................................................................................................... vi
List of Figures .......................................................................................................................... viii
Nomenclature .......................................................................................................................... x

CHAPTER 1 Introduction ......................................................................................................... 1
  1.1. Introduction and Motivation ......................................................................................... 1
  1.2. Aims and Objectives ................................................................................................. 2
  1.3. Summary of Approach ............................................................................................. 3
  1.4. Contributions ........................................................................................................... 4
  1.5. Thesis Organisation ................................................................................................. 4

CHAPTER 2 Literature Review: Automated Assessment of Code Scripts ......................... 7
  2.1. Introduction ............................................................................................................... 7
  2.2. Overview of Assessment in Education .................................................................... 7
    2.2.1. Assessment in Education .................................................................................. 8
    2.2.2. Assessment Approaches ............................................................................... 12
    2.2.3. Types of Programming Errors ........................................................................ 18
    2.2.4. Question Types in Assessment Approaches .................................................. 19
  2.3. Solutions for CBA ................................................................................................... 21
    2.3.1. Marking Systems for Code Scripts ................................................................. 21
    2.3.2. Clone Detection in CBA ............................................................................... 29
    2.3.3. Artificial Intelligence in CBA .......................................................................... 31
    2.3.4. Case-Based Reasoning (CBR) ...................................................................... 32
  2.4. Conclusion ............................................................................................................... 34

CHAPTER 3 Research Methodology ....................................................................................... 36
  3.1. Introduction ............................................................................................................... 36
  3.2. Research Onion ........................................................................................................ 36
  3.3. Research Philosophy ............................................................................................... 37
    3.3.1. Positivism ....................................................................................................... 37
    3.3.2. Realism .......................................................................................................... 38
    3.3.3. Interpretivism ................................................................................................. 38
    3.3.4. Pragmatism .................................................................................................... 38
    3.3.5. This Thesis’ Research Philosophy ................................................................. 39
  3.4. Research Approach .................................................................................................... 39
    3.4.1. Deductive ....................................................................................................... 39
    3.4.2. Inductive ........................................................................................................ 39
List of Tables

Table 2.1 Feedback qualities according to Mayer, 2004; Higgins and Bligh, 2006; Sadler, 1987; Wiggins, 2012 ............................................................... 18
Table 2.2 Types of Programming Errors ................................................................. 19
Table 2.3 Classification of Automatic Assessment Studies ........................................ 21
Table 3.1 Research Design Framework (modified by Vaishnavi and Kuechler, 2015) ...... 47
Table 4.1 Code Scripts ...................................................................................... 51
Table 4.2 Example of the blank line rule ............................................................... 60
Table 4.3 Example of the comment rule .............................................................. 61
Table 4.4 Example of the parentheses rule ........................................................... 61
Table 4.5 Example of the print message rule ....................................................... 62
Table 4.6 Example of the whitespace rule ......................................................... 62
Table 4.7 Example of the swap rule .................................................................. 63
Table 4.8 Experiment Results ........................................................................... 65
Table 4.9 Example of three different components .............................................. 67
Table 4.10 Control structure parts of components .............................................. 68
Table 4.11 Three different condition structures ................................................. 69
Table 4.12 Three different code segment structures ......................................... 70
Table 4.13 Similarity calculation between components ...................................... 71
Table 4.14 Examples of programming code ...................................................... 74
Table 5.1 Questions used to collect novice students’ solutions ......................... 82
Table 5.2 Numbers of preferred marking areas ............................................... 86
Table 5.3 Numbers of annotations used by the participants. .......................... 87
Table 5.4 Participants’ opinion about the most helpful marking area. ................. 89
Table 5.5 Participants’ opinion about the least helpful marking area. ................. 90
Table 5.6 Participants’ opinion about the necessary marking areas. ................. 91
Table 5.7 Participants’ opinion about the marking areas’ size ......................... 91
Table 5.8 Participants’ suggestions about the marking areas. ............................ 92
Table 5.9 Information on the provided comments according to marking areas. .... 93
Table 5.10 Data on the preferred marking areas. ............................................. 97
Table 5.11 Numbers of annotations used according to type. ........................... 98
Table 7.1 The participants’ responses on their paper-based assessment procedure 138
Table 7.2 The participants’ comments and opinions ......................................... 145
Table B.1 Participants’ reusable comments ......................................................... 176
List of Figures

Fig. 1.1 The structure of the thesis ............................................................................................................. 5
Fig. 2.1 Manual Assessment Process Cycle .............................................................................................. 13
Fig. 2.2 Bloom’s Taxonomy ...................................................................................................................... 17
Fig. 2.3 The CBR-cycle (adapted from Aamodt & Plaza, 1994) ................................................................. 34
Fig. 3.1 The research onion (Saunders at al. 2012) .................................................................................. 37
Fig. 3.2 The Framework of this research ................................................................................................... 48
Fig. 4.1 The proposed semi-automated assessment approach cycle ........................................................... 54
Fig. 4.2 Code Segment Type Hierarchy .................................................................................................... 56
Fig. 4.3 Control structure tree .................................................................................................................. 58
Fig. 4.4 Condition structure tree .............................................................................................................. 59
Fig. 4.5 Marking process model ............................................................................................................... 76
Fig. 5.1 Example of the marking page layout in the pilot experiment ....................................................... 84
Fig. 5.2 Example of the re-designed marking page layout ....................................................................... 96
Fig. 5.3 Participants response on the marking areas ................................................................................. 100
Fig. 5.4 Participants’ responses on the annotation types ......................................................................... 101
Fig. 5.5 Participants’ responses regarding the distance between the comment and the code segment. ................................................................................................................................................. 101
Fig. 5.6 Participants response on the length of the comment area ............................................................. 102
Fig. 5.7 Participants’ responses on the use of comments for similar code segments ................................. 103
Fig. 6.1 Overview of the marking system .................................................................................................. 109
Fig. 6.2 Full-marking process model ....................................................................................................... 113
Fig. 6.3 Partial-marking process model .................................................................................................... 116
Fig. 6.4 The case generation process ...................................................................................................... 117
Fig. 6.5 The case usage process .............................................................................................................. 118
Fig. 6.6 The full-marking tool user-interface ............................................................................................ 121
Fig. 6.7 A marked code segment in the full-marking tool ........................................................................ 122
Fig. 6.8 A marked code script in the full-marking tool ............................................................................. 122
Fig. 6.9 An example of a report on the marking process .......................................................................... 123
Fig. 6.10 The partial-marking tool (Part-I) user-interface ........................................................................ 124
Fig. 6.11 The partial-marking tool (part-II) user-interface ....................................................................... 125
Fig. 6.12 Accepted or modified comments in the partial-marking tool (part-II) user-interface .......... 125
Fig. 6.13 Annotation toolbar ................................................................................................................... 126
Fig. 6.14 Annotated code lines ................................................................................................................ 127
Fig. 7.1 Total time spent on each marking tool ................................................................. 133
Fig. 7.2 The average time spent for code segments on the full-marking tool ...................... 134
Fig. 7.3 The average time spent for code segments on Partial-Marking Tool ..................... 134
Fig. 7.4 Marking time spent on group members using the full-marking tool ....................... 135
Fig. 7.5 Marking time spent on group members using the partial-marking tool .................. 135
Fig. 7.6 Number of marked code segments according to type using the full-marking tool ....... 136
Fig. 7.7 Number of marked code segments according to type using the partial-marking tool .. 137
Fig. 7.8 Usability of the marking tools ............................................................................. 140
Fig. 7.9 Providing segmented marking in paper-based approaches .................................... 140
Fig. 7.10 The re-use comments in the marking process ...................................................... 141
Fig. 7.11 Using of tools in the participants’ institute ......................................................... 143
Fig. C.1 Marking time spent on Group 2 – full marking tool ............................................. 178
Fig. C.2 Marking time spent on Group 3 – full marking tool ............................................. 178
Fig. C.3 Marking time spent on Group 4 – full marking tool ............................................ 179
Fig. C.4 Marking time spent on Group 5 – full marking tool ............................................ 179
Fig. C.5 Marking time spent on Group 6 – full marking tool ............................................ 180
Fig. C.6 Marking time spent on Group 7 – full marking tool ............................................ 180
Fig. C.7 Marking time spent on Group 8 – full marking tool ............................................ 181
Fig. C.8 Marking time spent on Group 9 – full marking tool ............................................ 181
Fig. C.9 Marking time spent on Group 10 – full marking tool ........................................... 182
Fig. C.10 Marking time spent on Group 11 – full marking tool .......................................... 182
Fig. C.11 Marking time spent on Group 2 – partial marking tool ....................................... 184
Fig. C.12 Marking time spent on Group 3 – partial marking tool ....................................... 184
Fig. C.13 Marking time spent on Group 4 – partial marking tool ....................................... 185
Fig. C.14 Marking time spent on Group 5 – partial marking tool ....................................... 185
Fig. C.15 Marking time spent on Group 6 – partial marking tool ....................................... 186
Fig. C.16 Marking time spent on Group 7 – partial marking tool ....................................... 186
Fig. C.17 Marking time spent on Group 8 – partial marking tool ....................................... 187
Fig. C.18 Marking time spent on Group 9 – partial marking tool ....................................... 187
Fig. C.19 Marking time spent on Group 10 – partial marking tool ...................................... 188
Fig. C.20 Marking time spent on Group 11 – partial marking tool ...................................... 188
Nomenclature

Abbreviations

AST: Abstract Syntax Tree
CAA: Computer Aided assessment
CBA: Computer Based Assessment
CBR: Case-Based Reasoning
IDE: Integrated Development Environment
MC: Metric Comparison
SM: String Matching

Key terms

Case: A code segment (problem) and comment (solution) constitute a case.

Case-Base: Collection of marked code segments (cases) stored in a database.


Code Segment: Part of code scripts such as an else-if structure, for loop structure or while loop structure etc. A code script can consist of more than one code segment.

Comment: Comments help students improve their programming skills. They do not include any numerical grade.

Dynamic Assessment: Students’ program code is run and tested automatically, and the result of the programming code is subsequently checked to ascertain the correctness of the program.

Full-Marking Technique: A technique in which markers see the whole code script during the marking process and provide feedback for each code segment in a code script.
**Partial Marking Technique:** A technique in which markers see only a code segment instead of whole code script in the marking process.

**Segmented Marking:** Marking of each code segment in a code script.

**Semi-Automated Assessment Approach:** A hybrid of the full and paper-based assessment approaches.

**Static Assessment:** Students’ program code is tested and evaluated without running the program.
CHAPTER 1
Introduction

1.1. Introduction and Motivation

Programming languages are considered a core part of many courses/programmes in Higher Education (HE), such as the introductions to programming and object-oriented programming modules in computer science courses (Jenkins 2002; Lye and Koh, 2014; Trivodaliev et. al, 2017). Increasingly, a new trend has emerged where students from other departments such as physics, chemistry and mathematics have begun to learn programming languages (Parnas 1999; Brown et al., 2014). These students are considered novice programmers since they do not have any formal experience of programming and may struggle more with the basic elements of programming languages (Ichinco & Kelleher 2015). The assessment of student programming solutions (code scripts) often constitutes a significant workload for markers. Computer Aided Assessment (CAA) systems are used to reduce the workloads of markers, provide more consistent feedback and reduce human errors (Wang et al. 2011). Providing comprehensive feedback can be a complex process (Ala-Mutka 2005). CAA systems can adopt various approaches, including the fully automated and semi-automated approaches (Suleman 2008).

Fully automated assessment not only reduces markers’ workloads, but also provides more consistent feedback, because the programming code is tested automatically (Tiantian et al. 2007). A semi-automated assessment approach also helps markers to reduce the workload of the marking process. Furthermore, it may provide more comprehensive feedback by involving a human marker (Reininikainen 2006). According to Ala-Mutka (2005), fully automated assessment may ignore important parts of students’ solutions when providing feedback. Many authors agree that human markers need to be involved in the marking process since they can provide detailed comments on the required parts of students’ solutions (Jackson 2000; Reinikainen 2006). Furthermore, novice students need detailed feedback, which can be provided through a semi-automated assessment approach (Suleman 2008). This research therefore focuses on semi-automated assessment only.
Furthermore, many researchers are of the opinion that novice programmers should begin learning by providing solutions to short-answer questions, since they often lack the programming terminology to correctly describe a longer problem (Sheard et al. 2008). Moreover, Elva and Workman (2009) believe that novice students can achieve more programming experience through formative assessment (Keuning & Jeuring 2016).

This chapter provides an introduction to the thesis. It is organised as follows: Section 1.2 discusses the research aims and objectives and identifies the research questions. Section 1.3 discusses the approach adapted to meet these objectives and Section 1.4 outlines the research contributions. Finally, the organisation of the thesis is given in Section 1.5.

1.2. Aims and Objectives

The aim of this research is to develop a semi-automated assessment approach that assesses novice students’ programming solutions online and helps markers to provide personalised and consistent feedback. Many semi-automated assessment approaches have already been developed to assess novice students’ programming solutions. However, the literature review in Chapter 2 shows that the existing semi-automated approaches have not focused sufficiently on providing personalised, consistent and helpful (broad) feedback, nor do they adequately improve the efficiency of human markers in the marking systems to the point of justifying the widespread deployment of said systems.

The programming code assessment process can be time consuming for markers, since the marking part of the assessment process grows linearly with the number of student scripts marked (Wang et al., 2012). This research intends to reduce markers’ workloads and provide consistent feedback using a semi-automated assessment approach. This approach utilises the similarity of students’ solutions, since code segments (code snippets) from different code scripts are sometimes similar. If similar code segments receive the same comments from a marker, novice students receive consistent feedback. The research questions of this research therefore are:

1. Can a semi-automated assessment system be used to give personalised and segmented (structural) feedback?
2. Can a semi-automated assessment system be used to alleviate assessors’ workloads in terms of marking?
3. Can semi-automated assessment systems provide consistent feedback on students’ exercises?

The rest of this section describes the objectives of this research, which are:

1. To extract individual code segments from code scripts. Code scripts (programming solutions) are parsed based on the parsing criteria to obtain individual code segments such as ‘else-if’ or ‘while loop’ statements.
2. To find similar code segments amongst different code scripts. Similarity measurement formulas are used in order to achieve this task.
3. To develop a marking technique supporting the marker in providing feedback based on code segments.
4. To develop an approach that reuses a marker’s comment(s) for similar code segments in order to provide consistent feedback.
5. To carry out a feasibility study on the marking technique before developing the marking systems.
6. To develop prototype systems for both the full and partial marking techniques.
7. To evaluate both systems by user tests and compare their results.

The above objectives provided the inspiration for the research questions, which are:

1.3. Summary of Approach

The proposed assessment approach parses programming solutions to capture code segments. Generic (normalisation) rules are applied to the code segments to normalise them which increases the similarity between them. This approach then measures the similarity between code segments using a string match and groups them based on the similarity measurement values. A new marking technique (called segmented marking) is developed to mark the code segments which are grouped using the string match. Furthermore, in this research, the Case Based Reasoning (CBR) cycle is adopted to accelerate the automation in the marking process and enable the marker’s comments to be re-used for similar code segments. Thus, the proposed approach ensures the consistency of the comments generated during the marking process through the CBR cycle.

New full and partial marking models are developed based on the segmented marking concept. This research also develops prototypes of the full and partial marking tools which enable
marking of code segments in a code script. A feasibility study is conducted on the segmented marking technique in order to help identify the user interface design requirements for the full and partial marking tools. Finally, to assess the proposed approach, an evaluation study is performed on the implemented prototype full and partial marking tools.

1.4. Contributions

The contributions of this thesis are:

- Through the application of CBA in the assessment of programming solutions (code scripts), this research has contributed to an enhanced understanding of semi-automated assessment. In this context, human markers’ comments are re-used for similar code segments, which means that automation is made based on static assessment.

- A new semi-automated assessment framework is proposed consisting of an integration of a variety of methods. It can be used to enhance the automation of assessment of the programming solutions (code scripts).

- A novel marking technique (segmented marking) is developed based on the proposed assessment approach.

- Two novel marking process models are developed based on segmented marking, namely full and partial marking.

- A new case concept is defined based on segmented marking which enables the marking of a code segment using the CBR cycle. The CBR cycle contributes to ensuring the consistency and efficiency of feedback and also accelerates the automation in the marking process.

1.5. Thesis Organisation

Figure 1.1. depicts the structure of the thesis. Chapter 2 is divided into two main parts. The first part provides an overview of assessment methods in education whereas the second discusses solutions for Computer Based Assessment (CBA). Furthermore, the first part focuses on assessment purposes, methods and approaches and highlights the differences between the paper-based and CBA approaches. The types of programming errors and the question types in assessment approaches are also discussed. The second part reviews the potential full and semi-automated marking techniques. Furthermore, clone detection methods are examined to find the most suitable similarity measurement method. Finally, the role of the reasoning systems in
education is investigated to accelerate the marking process and provide consistent feedback based on segmented marking.
Chapter 3 initially discusses the research philosophies, approaches, methods and strategies, after which the appropriate choices for this research are presented. The overall research process is also presented to clarify the research design framework.

Chapter 4 introduces the proposed semi-automated assessment framework for programming solutions. This chapter focuses on the core part of the framework which contributes towards the marking of programming solutions based on the segmented marking technique. Generally, students’ solutions consist of similar code segments according to our observation. In such cases, similar code segments need to receive consistent feedback. Thus, the core part utilises various processes and presents a novel marking model adopting the CBR cycle.

Chapter 5 presents a feasibility study of the marking technique. This chapter’s aim is to not only measure feasibility of the marking technique but also identify the user-interface design requirements for the prototype marking tools.

Chapter 6 explains the design and development of the prototype marking tools (full and partial marking tools). Two marking models (full and partial) are developed based on the marking model presented in Chapter 4. The full and partial marking tools are developed based on the models.

Chapter 7 presents an evaluation study carried out using the prototype marking tools. It presents the study design, results (time saved in marking, quantify measurements such as the number of modified comments by the markers after the automated marking and the participants’ responses to the questionnaire) and key findings on using the tools.

Chapter 8 presents a summary of the thesis and draws conclusions from this research. It also summarises the research contributions, implications and limitations. Finally, it proposes a number of future directions to this research.
CHAPTER 2
Literature Review: Automated Assessment of Code Scripts

2.1. Introduction

The number of students learning programming languages has recently increased, especially in Higher Education (HE) (Brown et al., 2014). Traditional methods of assessment make it difficult to provide comprehensive, consistent, personalised and timely feedback for students’ solutions (Carter et al. 2003). Moreover, there is considerable pressure on HE to measure students’ learning and performance more formally (Lewis & Johnston 2002). Computer Aided Assessment (CAA) is an important field to address these issues. CAA is a term used for assessment processes using computer technology (Repository 2010). It refers to the assessment of students’ abilities using a computer and is usually used for marking (Conole & Warburton 2005b). However, CAA is currently being superseded by Computer Based Assessment (CBA). The reason is that CAA uses computers for assessing students learning (McKenna, 2004), while in contrast, CBA uses computers for the entire assessment including delivery of feedback, management of assessment and administration of assessment (King, 1994). Researchers are focusing on a variety of subjects for numerous types of assessment in HE based on CBA.

This chapter presents a literature review for the research conducted in this project. The review focuses on two main aspects, namely an overview of assessment methods in education and the solutions provided by CBA. The first section discusses the assessment types, methods and highlights the importance of CBA marking. However, due to the importance of the marking process for this research, prominent CBA solutions are examined in a separate section.

2.2. Overview of Assessment in Education

This section starts with an examination of assessment types and methods and their applicability to given contexts as well as the advantages and disadvantages of those methods (Section 2.2.1). The examination of assessment types and methods could be helpful for markers in deciding the most beneficial assessment type and method for novice programmers. Section 2.2.2 then provides information on assessment approaches including paper-based (the more
traditional way of marking) and CBA, where the discussion highlights how CBA approaches can be further improved to overcome the issues of the paper-based approach and provide quality feedback. Feedback also needs to take into account programming errors (Singh et al. 2013), and as such, the types of programming errors are presented in Section 2.2.3. Finally, questions types are described in terms of CBA based on Bloom’s Taxonomy in order to highlight which types of questions can be most helpful in improving novice programmers’ programming skills (Section 2.2.4). The reason is that programmers cannot improve their own programming skills sufficiently if they provide solutions for low level questions such as multiple-choice questions. However, they can improve their programming skills significantly if they practice writing code and then receive quality feedback on their work.

**2.2.1. Assessment in Education**

Brown and Knight (1994) believe that assessment is an important part of the learning experience. It complements the learning process and provides students with feedback on their learning. The skill development process of students is reported through assessment (Harlen et al., 1992). Markers capture information about each student’s progress, and so can develop the teaching and learning experience through assessment (Taras, 2005). In addition, Bull and McKenna (2004) state that assessment methods should be correctly chosen by tutors or markers in order to support students sufficiently. In this respect, this section provides a discussion of the most prominent assessment purposes and methods.

**2.2.1.1. Assessment Purposes**

There are three types of assessment, namely formative, diagnostic and summative. Formative and diagnostic assessments have the same purpose, which is to improve the learning experience (QAA, 2014). Summative assessment aims to measure students’ understanding of a subject. Assessment types are therefore quite important for providing information about students’ progress and achievement. They are described here based on their advantages and disadvantages. In addition, this research’s assessment purpose is described at the end of this section.
a) Formative Assessment

For an assessment to be formative, it should provide feedback that refers to the existence of a gap between the students’ real levels and the required standard. It should also give students guidance to improve their work and reach the required standard. Each student’s level can be discussed and suitable next steps may be designed or planned through formative assessment (Ramaprasad, 1983; Sadler, 1998; Berry and Meekings, 1985; Watson, 2008). Bull and McKenna (2004) state that formative assessment is directly related to the enhancement of student’s education based on feedback. Students may understand their own learning more deeply through formative assessment (Clark, 2011). Bedford and Price (2007) also highlight that useful comments in the feedback process should be related to each student’s learning and understanding, rather than based only on numerical marking. Knight (2001) specifies that formative assessment provides personal comments in the feedback for each student. Pop quizzes, tutorial exercises and lab work are some examples of formative assessment. Formative assessment may also cause some problems (Crossouard and Pryor, 2012). For instances, markers/tutors need to engage students with new ideas based on the assessment results, as otherwise, students cannot improve own programming skills adequately. Thus, feedback should focus on the gap between the students’ current learning level and the required standard. Shepard (2005) believes that formative assessment is quite important if it is the preferred assessment method during the term and the marker should provide sufficient comments and assist students with useful exercises.

b) Diagnostic Assessment

Students’ prior knowledge is determined through diagnostic assessment which has the same purpose as formative assessment (Brown and Knight, 1994). Diagnostic assessment is used by markers to define students’ knowledge before a course of study starts (Sclater and Howie, 2003). Dixson and Worrell, (2016) specify that markers should capture information about students’ strengths and weaknesses so they can plan what topics should be taught and how they should teach them. Diagnostic assessment can also analyse students’ knowledge, skills weaknesses and difficulties in understanding (Conole and Warburton, 2005). Pre-tests, self-assessments and interviews are all types of diagnostic assessment. However, these types of assessments may cause some problems if they are not applied successfully (Grissino-Mayer, 2001). According to Reynolds (2004), diagnostic assessment may consist of key information relating to students’ knowledge on a subject. In this case, educators need to create a useful and structured plan for each student to develop his/her skills (Higgins and Bligh, 2006). However,
feedback is still important since tutors can compare and monitor the students’ level during the term based on the feedback.

c) **Summative Assessment**

Summative assessment measures students’ learning and reports their achievements at the end of the learning period (Scriven, 1967). Chalmers and McAusland (2002) indicate that students are informed of exams or coursework in advance so they can prepare for them, because a summative assessment is an official assessment. Summative assessments include final exams, midterm exams and projects. The purpose of summative assessment is to provide an end grade. However, summative assessments should also have the following qualities (Assessment Reform Group, 2002):

- **Validity**: the assessment should cover students’ achievement in terms of special purpose.
- **Reliability**: results should be consistent.

Shepard (2005) state that the validity and reliability qualities are important for both students and markers. Feedback is also important since students should be informed about their weaknesses (Ala-Mutka, 2005). In addition, markers can compare the students’ level using the feedback from the diagnostic assessment to outcomes of the summative assessment. Thus, students can see their improvement during the term.

d) **Assessment Purpose for Novice Programmers**

In Section 2.2.1.1, assessment types are compared based on their advantages and disadvantages and the assessment type addressed by this thesis is explored. This research aims to provide feedback for novice programmers. In this context, feedback plays a vital role in the learning experience. Formative assessment methods generally provide targeted feedback to students, rather than the final results of their study (Gütl, 2008). Feedback should be made useful for students; as otherwise, it cannot help them to develop their programming skills. If feedback is comprehensive and of sufficient quality, students can understand and address their mistakes more efficiently (Bull and McKenna, 2004; Crooks, 1998). Moreover, students can further improve their programming skills through the feedback during term time. Therefore, formative assessment is considered to be the preferred type of assessment for this research.
2.2.1.2. **Assessment Methods**

In literature, two main methods of assessment are described, namely oral and written assessment methods. This section discusses both methods to decide which is the most beneficial for students (especially for novice programmers) in terms of providing solutions for questions, and for markers in terms of providing consistent and efficient feedback.

**a) Oral Method**

Oral assessment is a well-known assessment method where the marker asks questions verbally to students and requests responses from them (Ayesha et al., 1999). In this method, students provide information about what they have learned about the topic or subject (Pollitt and Ahmed, 1999). However, the behaviour of the marker can positively or negatively affect the student. The student's motivation may be weakened and they may not be able to respond adequately if they are negatively affected by the marker’s behaviour (Kahneman, 2011).

**b) Written Method**

Written assessment is based on students reading and then writing their answers to written questions (Schuwirth and Van Der Vleuten, 2004). Students’ answers can be partially correct for complex question types. In this sense, the marker can provide more detailed feedback based on the student’s solution (Dijkstra et al., 2010). Written assessment methods include multiple choice questions, short answer questions and true/false questions (Thelwall, 2000). Written assessment methods can be applied through paper-based assessments or computer-based assessments (CBA).

Brown et al. (2014) state that written assessment methods can be more effective than oral assessment methods in essay type exercises (such as programming exercises). Within a written assessment, students have more time to answer questions compared to oral assessment methods. Furthermore, all students receive the same questions and markers can provide consistent feedback (Harris and Brown, 2013). The marker can provide feedback easily if a question requires a short answer and the number of students in the classroom is not high. Question types are described in Section 2.2.4. Furthermore, CBA systems contribute towards providing feedback in a short time (Pettit et al. 2015). In addition, questions need to be well prepared to make them easily understandable by students. If students understand questions well, they are in a better position to provide correct solutions. In this sense, markers can also provide more consistent feedback (Abulencia, 2016).
2.2.2. Assessment Approaches

Markers may assess students’ programming code through different assessment approaches including paper-based assessment (traditional way of assessment) and Computer-Based Assessment (CBA) (Cheang et al., 2003). Paper-based assessment approaches have the longest history. This section provides an explanation for the paper-based assessment (manual marking) cycle. Paper-based assessment approaches have some disadvantages compared to the CBA approach, which are highlighted in Section 2.2.2.1. Many markers prefer CBA approaches due to the disadvantages associated with paper-based assessment approach (Seale 2002).

2.2.2.1. Paper Based Assessment

Students’ programming codes were mostly assessed in paper-based methods before 1960 (Cheang et al., 2003). Paper-based assessment has some advantages and disadvantages. One advantage of paper-based assessment is that each student’s strengths and weaknesses are highlighted by the marker (Ihantola et al., 2010). The marker may also provide detailed feedback if they deal with several parts of the programming code. On the other hand, one disadvantage of paper-based assessment is that markers could potentially make mistakes in their assessment. Programming code assessment is not an easy task and can be time-consuming for markers, who may lack consistency during the marking process (Lewis & Johnston 2002). Carter et al. (2003) also state that the assessment of students’ programming codes in an overcrowded classroom is a major issue for markers, which could often lead to fewer assessment tasks set by markers. Therefore, many researchers prefer the use of computers in programming assessment (Ala-Mutka, 2005).

As highlighted in Section 2.2.2, paper based assessment is also referred to as manual assessment. In the manual assessment process cycle, the marker has to complete several processes (Brown et al. 2004). The assessor initially prepares a question and then present it to students to obtain their solutions. Subsequently, the solutions are marked by the marker. Finally, students get feedback on their solutions. In this sense, programming assessment consists of four processes: question preparation, obtaining program code, marking and grading and feedback generation. Figure 2.1 illustrates the traditional assessment cycle, which is created based on the aforementioned four processes.
In **Process 1**, the assessor prepares questions and then provides model answers, rubric or marking scheme for the prepared questions. The assessor highlights important parts in the question, and the model answer, rubric or marking scheme are created based on the highlighted important parts.

**Challenges in Process 1**

The questions need to be well-defined which means that they should cover important parts of the subject. The parts are quite important for students, since they provide their own solutions based on the parts (Miriylala and Harandi, 1991). However, even if preparing a well-defined question is a challenge for the assessor, it can be beneficial for students since they may understand the allocation of marks to content better.

In formative assessment, higher questions levels are especially important. Questions levels are discussed in Section 2.2.2.2. Novice programmers need to provide programming code for low-level questions to gather experience, as otherwise, they may struggle with providing programming code for high-level questions. Assessment is part of the learning process and students learn from their mistakes (Clark 2011). Formative assessment aims to provide feedback, which should support the students in terms of developing their own programming skills (Watson 2008). It also helps assessors to obtain information about students’ levels. In this case, the assessor prepares questions based on the students’ levels and the questions level is subsequently increased according to the development progress of the students’ programming skills (McAlpine
2002). Therefore, the assessor must write different well-defined questions according to the students’ levels, which is the main challenge of this process.

Short questions (scenario) are used in tutorials in order to set programming exercises for students (novice programmers). The assessor can adjust the complexity of the question by asking additional questions based on the students’ knowledge whenever it is deemed necessary. A common problem in tutorials is that students may sometimes hesitate to ask questions (Low et al, 2009), which prevents them from obtaining useful knowledge and feedback. However, if students provide solution separately (i.e. individually), the assessor can check the solutions and provide detailed feedback, thus alleviating the aforementioned problem. Since feedback based on formative assessment is very important, especially for novice programmers, question preparation is particularly critical in the context of programming.

- **In Process 2**, students provide solutions to the questions. Students may provide their own programming solutions on paper. They can also use different types of IDE, such as generic IDEs or special ones. IDE means a source code editor. In this process, obtaining students’ solutions is more important than the type of IDE. However, the IDE is extremely important in terms of obtaining well-formed solutions. Students may also submit their programs through online or offline submissions. Online submission means electronic submission via e-mail or the internet (Clark and Baillie-de 2007), while offline submission means the students provide the solutions without using the internet, via USB drive etc., (Shortis and Burrows 2009).

**Challenges in Process 2**
Students sometimes provide mal-formed/illegible solutions (programming codes) if they use paper (Thomas, 2004).

- **In Process 3**, the marker marks students’ solutions according to a model answer, rubric or marking scheme (Ahoniemi and Reinikainen 2006). The marker’s workload depends on the number of students in the classroom. If the number is high, the marker’s workload increases, which also affects the feedback consistency (Suleman 2008). In this case, markers might try to ask short questions which require short answers. In this way, their workload is reduced, which could also impact the feedback consistency.
**Challenges in Process 3**

The main task of this process is marking students’ programming solutions. Markers may use different methods of marking and feedback provision for the students’ programming solutions. However, students’ solutions are naturally different from each other in terms of syntax, even if they result in the same outcome. As such, markers need to provide a variety of model answers. This is extra workload for markers because they must match certain parts of student solutions with model answers, which is the main challenge of this process. In this case, a marker may prefer to provide feedback using a generic model answer. Furthermore, the marker might also provide superficial and inconsistent feedback, since students’ programming codes are marked manually (Cheang et al. 2003). Challenges to the marking process can therefore be summarised as follows:

- Matching of student solutions with model answers creates an extra workload for markers.
- Providing consistent feedback.
- Providing high quality feedback. This is because feedback quality depends on the marker’s experience. If the marker is not experienced, students often do not receive high quality feedback.
- Marking load.

**In Process 4**, students receive feedback, including comments and/or grades. Markers may provide feedback in a variety of ways. For example, they may provide general feedback for a crowded classroom. According to Ala-Mutka (2005), feedback is more important than numerical grades in formative assessment.

**Challenges in Process 4**

Students’ programming solutions may contain similar mistakes; therefore, markers need to provide the same comments to provide consistent feedback. Furthermore, the feedback should be clear, which will help students in terms of understanding their mistakes.

In each process in Figure 2.1., the marker’s workload is proportional to the population of the classroom. A crowded classroom mainly affects the marker’s workload in the marking process. Hence, the efficiency of the marking process is important. Many researchers have focused on
the marking process in online programming assessments, which are discussed in Section 2.3. However, researchers utilise different processes to those outlined Figure 2.1 to make the assessment automatic. Some researchers have opted for automating Process 2 (Obtaining Programming Code) (Staubitz et al, 2015) while some researchers have automated Process 3 (Marking and Grading) (Kitaya and Inoue, 2016). Furthermore, some researchers have automated Process 4 (feedback generation) which also means that the feedback delivery is provided online (Singh et al, 2013). This research focuses on the automation of Process 3, with further details presented in Section 2.3.

2.2.2.2. Computer Based Assessment

Computer Based Assessment (CBA) has been applied since 1960 (Hollingsworth, 1960). Many assessment systems in computer science were initially developed before other assessment systems in different areas such as in chemistry, mathematics and physics were developed. CBA has also used to assess different subjects in many universities since the 1990s (Stephens et al., 1998).

CBA systems have generally been developed to capture and mark students’ solutions. CBA systems follow two main different approaches, namely fully-automated assessment approach (Saikkonen et al. 2001) and semi-automated assessment approach (Jackson et al. 2000). Weinberger (2011) state that in fully automated assessment approaches, students submit their assignments and these submissions are subsequently assessed purely by a computer. In contrast, in semi-automatic assessment approaches, computers help the human marker in the marking process. Weinberger (2011) also highlight that the marker (human) is important in terms of feedback in semi-automatic assessment. In computer science departments, semi-automatic assessment is mostly used for lab and small class exams as a form of formative assessment to improve students’ programming skills (Gordon, 2012). Tshibalo (2007) also state that since markers’ workloads have increased, using CBA may result in reducing their workloads.

Markers should decide on an approach according to what they want to assess for developing students’ skills. Bloom (1956) developed a taxonomy which is referred to as Bloom’s Taxonomy and is illustrated in Figure 2.2. It has six levels from lower (knowledge) level to higher (evaluation) level.
Lower levels are related to basic learning approaches. For example, the knowledge level can be evaluated in assessment through the use of multiple choice questions. Measuring students’ aptitude becomes more difficult as you move up through the levels as higher levels are related to deep learning. Students use high cognitive skills like problem solving and the ability to analyze in order to construct long-term understanding in higher levels (Forehand, 2010).

Fully-automated marking can be performed at the lower levels of Bloom’s taxonomy because these levels require one correct answer such as the case with multiple choice questions (Clark & Education 2002). However, full-automated assessment for higher levels cannot be applied easily because student solutions are generally written answers including programming codes or essays (Pillay & Farquharson 2014). In this case, a computer supported approach (semi-automated assessment) needs to be employed to assess students’ solutions. Furthermore, questions types are quite important for deciding which type of CBA needs to be used to provide feedback (Bull and McKenna, 2003). Detailed information about the question types on CBA approaches is presented in Section 2.2.4.

Many researchers agree that assessment qualities are important in terms of developing students’ skills (Mayer 2004; Wiggins 2012; Sadler 1989; Higgins and Bligh, 2006). Feedback quality is considered an important factor in developing students’ skills. Furthermore, feedback helps students to understand problems and find suitable routes towards addressing them. Table 2.1 shows the feedback qualities according to researchers’ comments.

![Fig. 2.2 Bloom’s Taxonomy](image-url)
CBA systems use different types of assessments including summative and formative assessments, which have been discussed in Section 2.2.1. Quality feedback needs to provide helpful hints for students to improve their programming skills (Altadmri and Brown, 2015). Thus, the next section discusses the types of programming errors.

### 2.2.3. Types of Programming Errors

Many programmers (especially novice programmers) encounter programming errors (Hristova et al, 2003). However, CBA systems mostly provide limited feedback on programming errors, which are discussed in detail in Section 2.3. Novice programmers need to receive helpful comments on programming errors to improve their solutions and programming skills (Truong et al, 2004). This section provides information on the types of programming errors. According to Ebrahimi (1994), programming errors contribute to students’ problem-solving strategies. Programming errors are classified into three common types, namely syntax, semantic and logic errors (Youngs, 1974; Kaczmarczyk et al., 2010). Table 2.2. provides definition for each type of programming errors.

**Table 2.1 Feedback qualities according to Mayer, 2004; Higgins and Bligh, 2006; Sadler 1987; Wiggins 2012**

<table>
<thead>
<tr>
<th>Quality</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meaningful</td>
<td>Enhance learning and improve student achievement</td>
</tr>
<tr>
<td>Timely</td>
<td>Effective for student learning</td>
</tr>
<tr>
<td>Sufficient</td>
<td>Useful and descriptive for student</td>
</tr>
<tr>
<td>Reliable</td>
<td>Promote consistency between assessors</td>
</tr>
<tr>
<td>Efficient</td>
<td>Be manageable within the constraints of resources</td>
</tr>
<tr>
<td>Formative</td>
<td>Give many opportunities to learn through feedback</td>
</tr>
</tbody>
</table>

CBA systems use different types of assessments including summative and formative assessments, which have been discussed in Section 2.2.1. Quality feedback needs to provide helpful hints for students to improve their programming skills (Altadmri and Brown, 2015). Thus, the next section discusses the types of programming errors.

### 2.2.3. Types of Programming Errors

Many programmers (especially novice programmers) encounter programming errors (Hristova et al, 2003). However, CBA systems mostly provide limited feedback on programming errors, which are discussed in detail in Section 2.3. Novice programmers need to receive helpful comments on programming errors to improve their solutions and programming skills (Truong et al, 2004). This section provides information on the types of programming errors. According to Ebrahimi (1994), programming errors contribute to students’ problem-solving strategies. Programming errors are classified into three common types, namely syntax, semantic and logic errors (Youngs, 1974; Kaczmarczyk et al., 2010). Table 2.2. provides definition for each type of programming errors.
Table 2.2 Types of Programming Errors

<table>
<thead>
<tr>
<th>Type of Error</th>
<th>Definition</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>Syntax errors represent incorrect statements in a programming language Jackson et al. (2000).</td>
<td>Spelling, punctuation errors, else without if etc.</td>
</tr>
<tr>
<td>Semantic</td>
<td>Semantic errors refer to the meanings of programs, and are also concerned with the interpretation of the expression of programs (David, 2012).</td>
<td>File not found, type mismatch, using a non-initialised variable (the most common error) etc.</td>
</tr>
<tr>
<td>Logic</td>
<td>A program may run correctly but produce incorrect results due to poor understanding of the problem (Spohrer and Soloway, 1986).</td>
<td>Incorrect boundary conditions, incorrect constants etc.</td>
</tr>
</tbody>
</table>

Hristova et al., (2003) stated that syntax errors are fixed easily compared to semantic and logic type errors. According to Kaczmarczyk et al. (2010), many marking tools cannot provide feedback on logic type errors due to the fact that the programming code runs correctly and provides outputs regardless of logic errors (Ebrahimi 1994). In this case, students (especially novices) should receive comments on each error type, especially logic type errors as feedback (McIver, 2000). If the feedback is sufficient and supportive, the students’ learning experience can be made more efficient, as highlighted in Section 2.2.1.1.

2.2.4. Question Types in Assessment Approaches

Objective and free-response question types are used in educational assessments (Jordan 2013). Objective questions refer to lower levels in Blooms’ Taxonomy, which was discussed in Section 2.2.2. The levels relate to a single correct answer such as true/false or multiple-choice questions. On the other hand, free-response questions refer to higher levels than objective question types, namely program code solutions and diagrams etc. This section discusses these types of questions in the context of CBA.

2.2.4.1. Objective Questions

One advantage of objective questions is their suitability for CBA. However, these types of questions refer to low-level question types in Bloom’s taxonomy. Many examiners agree that objective questions can be asked to develop student skills (Testing 2002; Carcinoid et al. 2010).
According to national survey results, 84% of CBA approaches use objective questions (Higgins and Bligh, 2006). Furthermore, the most popular objective question type are multiple choice questions (Warburton & Conole, 2003).

2.2.4.2. Free Response Questions

Free-response question types are more complex question types than objective questions, as they refer to the higher levels of the Bloom’s taxonomy (Davies 2001; Conole & Warburton 2005b). Many studies have been developed for automatic marking based on free-response questions including essays, diagram type solutions and computer programming codes. Many researchers agree that students cannot sufficiently improve their skills based on objective questions alone (Bible et al, 2008; Simkin et al, 2011). The reason is that students choose only one correct answer in objective questions. In contrast, in free response questions, students think more deeply when they provide their answers and can often provide different but equally correct solutions (Kitchenham, 2006). In this sense, markers can also provide more detailed and personalised feedback for each student. For these reasons, free response questions are used in this research.

Section 2.2. presented an overview of assessment in education. Formative assessment is chosen for this research since novice programmers need to regularly receive feedback during term time in order to improve their programming skills. In addition, written assessment is chosen for this research since students have more time to provide their solutions compared to the oral assessment method. Section 2.2 also discussed paper-based assessment and CBA and their advantages and disadvantages were highlighted in Section 2.2.2. This research focuses on semi-automated assessment approach in order to overcome the disadvantages of the paper-based and the full-automated assessment approaches. The types of programming errors were also presented (in Table 2.2) since high quality feedback should cover programming errors. Finally, the use of free response questions is selected for this research as described in Section 2.2.4.

As highlighted in Section 2.2.1., researchers have focussed on different processes of the manual assessment cycle (Figure 2.1.). This research aims to automate the marking process (Process 3 in Figure 2.1) of the manual assessment as much as possible, for the reason that the main issue of manual assessment is with regard to providing consistent feedback.
2.3. Solutions for CBA

Marking of novice programmers’ solutions is important in this research for providing high quality, consistent and personalised feedback. Details of the provided feedback based on the marking systems and the efficiency (marking time and workload) and effectiveness (accuracy and consistency) of the systems are described in Section 2.3.1. The systems focus on dynamic assessment, static assessment or both. In dynamic assessment, students’ program code scripts are run and tested automatically, and the result of the programming code scripts are subsequently checked to ascertain the correctness of the program. By contrast, in static assessment, students’ program code scripts are tested and evaluated without running the scripts (Sharma et al., 2014). This research proposes a semi-automated assessment approach and utilises the similarity of students’ solutions, as described in Section 1.2. Therefore, the usability of the similarity measurement (clone detection) techniques in CBA systems are investigated in Section 2.3.2 to decide the most effective technique. In addition, Artificial Intelligence (AI) approaches can be preferred based on the purposes of providing feedback in education. Thus, reasoning systems in AI are described in Section 2.3.3 and detailed information is also provided on the Case-Based Reasoning (CBR) cycle in Section 2.3.4.

2.3.1. Marking Systems for Code Scripts

Many markers prefer paper-based assessment in programming exercises. Staubitz et al. (2014) state that most markers (74%) preferred lab exercises of manual assessment. However, the paper-based assessment of computer programming is inefficient and it presents an increased workload for the marker when the number of students in the class is high (Kolb, 2014). CBA systems could help markers to accurately estimate the programmer’s knowledge. Table 2.3 presents the classification of automatic assessment.

<table>
<thead>
<tr>
<th>Assessment Approach</th>
<th>Purposes of Assessment</th>
<th>Types of Programming Errors</th>
<th>Types of Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3. Summative</td>
<td>3. Logic</td>
<td></td>
</tr>
</tbody>
</table>
The fully-automated and semi-automated systems have crucial differences. Suleman (2008) listed the main differences between the full-automatic and semi-automatic systems:

- The marking process is completed by a marker, using a semi-automated approach.
- In the semi-automated approach, markers can choose static or dynamic testing. If a marker chooses static testing, novice programmers’ code scripts do not need to produce an output, because markers can provide feedback even if the code scripts are not completely correct.
- A semi-automated assessment system can provide more helpful feedback through human markers than a fully automated system can.

The rest of this section presents a discussion of the fully-automated and semi-automated marking systems based on their marking techniques.

2.3.1.1. Fully-Automated Marking Systems

Fully-automated assessment systems utilise dynamic assessment since human markers do not provide any feedback directly in these systems. Some of the marking systems provide feedback regarding not only syntax errors but also logic errors. The detail of the logic errors in dynamic assessment depends entirely on the execution file (Kaczmarczyk et al., 2010). This section describes the prominent marking approaches of the fully-automated assessment of students’ programming solutions. Even if many fully-automated marking approaches are provided for only non-novice programmers, the prominent marking approaches are provided for not only non-novice programmers but also novice programmers. Thus, they were chosen for this research.

1. Scheme-Robo is a system developed by Saikkonen, Malmi and Korhonen (2001). Students use the system for two types of questions which are ‘fill-in-the-gap’ and code writing. In this system, students can resubmit their own assignments a certain number of times. The output of the student’s solution and the model answer are compared to measure the correctness of the solution, which is a process that is part of dynamic assessment. Also, this system executes the student’s solution to provide feedback on its general structure. It is developed based on the summative and formative assessment. However, this system focuses on the entire code structure instead of specific parts of the code. In this case, this system may provide superficial feedback even if it is able to provide consistent feedback. Finally, although this system provides syntax and logic
types of programming errors, the feedback quality depends on the quality of the executed file. ‘Fill-in-the-gap’ type of questions can be very helpful for novice programmers and are often used in order to improve the students’ programming skills before they move to write a code.

2. Marmoset monitors students’ progress and sends feedback to both the student and the examiner (Spacco et al. 2006). It was developed for novice programmers only. Students can submit their projects whenever they need feedback. Marmoset provides separate feedback for each submission based on dynamic assessment and is developed based on formative assessment. If a student’s code script passes all public tests, their project is accepted as correct. Furthermore, Marmoset allows the marker to check students’ progress. The system provides textual (comment) and numerical feedback if a student’s solution does not contain any syntax errors. This system is developed for novice programmers who may struggle when attempting to fix syntax errors. However, a drawback of this system is that it only presents syntax errors instead of a detailed explanation on how novice programmers can fix the errors. In this sense, this system could provide tips to fix the syntax errors of novice programmers. Finally, Marmoset also provides feedback on logic type programming errors if the program is executed. In addition, students can get helpful feedback from the marker since marmoset allows the marker to check students’ progress. This allows the marker to provide some tips and comments for students in order to improve their programming skills.

3. In the system developed by Wang et al. (2007), the structure of the student program and a model answer are matched using semantic structure similarity based on dynamic assessment, which is a general automatic grading approach. In this approach, students’ programs are standardised using specific rules. Such rules are created within the system to standardise students’ program code. The aim of this standardisation is to eliminate syntactic variations, and so reduce the number of model answers, improving the program matching of the system. Furthermore, variables are renamed, redundant statements are removed and statements are reordered to improve the automatic matching by the system. Lastly, the semantic structures of the student programs and the model answers are matched, and the students subsequently receive their grades. This system is developed based on the summative and formative assessment. It also provides comments about syntax and logic type errors. In addition, removal of redundant parts may increase the similarity between code scripts and as such, this system may not need too many model answers to provide feedback. Furthermore, markers can focus on
improving the quality of the execution file in order to provide better feedback instead of providing model answers. However, students’ solution can be partially correct and may not match with model answers. In such cases, students may receive incorrect or insufficient feedback.

4. In the system of Kitaya and Inoue (2014), students receive syntax errors if their program does not correctly compile. These errors are saved by the system to display to the student. The student can fix these errors and then re-compile the code to obtain the result of his or her program. The system executes both the student’s answers and the model answers, and then compares their results to measure the correctness of the student’s answers and provides feedback based on dynamic assessment. In this sense, the saved syntax errors can help students to improve their programming skills since novice programmers mostly make syntax errors. However, the assessor needs to provide all possible model answers to automatically mark the students’ programming codes. This can lead to extra workload for the marker. This system is developed for the purposes of formative assessment and also provides feedback on logic errors if students’ solutions are executed.

The described fully-automated marking systems are useful in terms of providing instant and consistent feedback. However, the systems must focus on certain or specific parts of students’ solution to provide helpful and comprehensive feedback. In this sense, developers need to extend the tools to provide better feedback. However, such a task would be time consuming. Many researchers state that marking tools should enable human markers to make the feedback more detailed (Robinson and Carroll, 2016). Therefore, semi-automated marking systems are examined in following section.

2.3.1.2. Semi-Automated Marking Systems

Although many CBA marking systems based on the fully-automated assessment approach have been designed, CBA marking systems based on the semi-automated assessment are not as common as fully-automated marking systems. The prominent semi-automated marking systems are described in this section. Some marking systems are developed for only expert programmers, while others are also developed for novice programmers. Furthermore, while each of these systems utilise formative assessment, some also incorporate summative assessment.
1. In TRY (Reek, 1989), the students submit their programs and the marker tests them. If the marker feels that more comments on the students’ programming code are required, he or she can extend the feedback. In TRY, several testing approaches were tried. Initially, the student’s and model answer’s output are character-by-character matched, which is a process that is part of the dynamic assessment. If they match, the student’s answer is correctly accepted by the system. In the second approach, the student’s answer is normalised before matching with the model answer, where the redundant parts of student’s answer are removed. The latter approach is one in which students must print the output based on the marker’s definition in the question. Additionally, students’ code scripts must pass all tests to be considered successful. The TRY system then provides feedback on which scripts pass or fail. In addition, TRY also provides feedback on syntax and logic errors. Finally, the marker can provide comments on the student’s answer based on the static assessment. In this system, the second approach can be especially helpful for markers since the marker does not need to provide many model answers. Furthermore, the number of similar code script can increase after the applying the second approach which may increase the feedback consistency. The TRY system has some drawbacks including:

- The output of students’ programming code and the model answer are character-by-character matched in this approach. In this case, although the student answer may be correct, it may be different in terms of syntax/format to the model answers. In this case, the student answer is deemed incorrect. Thus, in such cases, the student could receive incorrect feedback.

- The addition of further comments by the marker to extend the feedback provided by the system could be time consuming. Furthermore, the marker may also provide inconsistent comments.

2. Jackson (2000) developed a semi-automatic assessment system. The system applies software metrics (e.g. level of commenting, use of indentation, etc.) to students’ programs. It assesses both syntax and logic errors. These errors are assessed automatically by configuration files created by the markers utilising dynamic assessment. In addition, the markers support the system in providing quality feedback. The system asks the marker questions regarding students’ program code as part of the assessment process. In this case, comments are suggested to the marker in the testing phase. The marker chooses any appropriate suggested comments to complete the
marking process. This system cannot provide any feedback without the interference of a human marker. If useful and helpful comments are suggested to the marker in the marking process, the system can provide helpful feedback. However, this system also has some drawbacks such as:

- Markers cannot write their own comments in the marking process. They can only choose from the default comments provided before the marking process. In this sense, the feedback cannot be easily modified during the marking process.
- The human marker must choose a comment for each solution in the marking process. This can be a repetitive and time-consuming process.

3. Ala-Mutka and Järvinen’s system (2004) assesses program compilation and dynamic behaviours (outputs and efficiency). After students complete an assignment, they submit it. If the submitted program script runs, the system gives feedback on the correctness of the solution and the logic errors. However, if the program does not run, the program automatically provides a list of the identified syntax errors. The marker can modify the feedback after the system gives it, that is, he or she can add information or delete unnecessary parts of the feedback based on the static assessment. Thus, the system can provide sufficient feedback. In addition, the system provides feedback on syntax errors even if the students’ solutions do not run. Furthermore, the system could provide only a comment (e.g. “your solution is incorrect”) instead of listing the syntax errors. This could be troublesome for the novice programmers since they usually have little or no experience on how to fix the syntax errors. The system also faces some issues such as:

- Modifying the feedback for each student’s code script could be repetitive and time consuming for the marker.
- The marker may provide inconsistent feedback while reviewing the automatically provided feedback. This is because the marker provides feedback based on static assessment of each code script, which may result in providing inconstant feedback.

4. ALOHA (Ahoniemi and Reinikainen, 2006) divides a given assessment into smaller parts. The marker can concentrate on a single aspect of the work instead of giving a general grade for the final work. ALOHA provides rubric online in a way that each marker must fill them correctly and in a similar way. An assessment rubric is a guide
listing specific criteria for grading smaller parts in the code script. The marker can choose a ready-made comment to be added for the student’s feedback. ALOHA provides only text-based feedback and utilises formative and static assessments. In this sense, the use of a rubric can help the marker if it is created in great detail. That is, if the marker makes comments on smaller parts using the detailed rubric, they can provide strong and comprehensive feedback for novice programmers. However, ALOHA has some drawback including:

- The marker should provide feedback according to a rubric. The marker cannot make comments outside of the rubric. If a rubric is created superficially, the marker may not be able to provide helpful comments.
- The marker must mark all programming code respectively. It is a workload for the marker and could lead to inconsistent feedback.

5. Sakai (Suleman, 2008) can compile, test, execute and score a student’s program without human intervention. It is developed based on dynamic and static assessments. If the output is incorrect, feedback is then given to the student regarding his/her produced output and the expected one by the human marker. If the output is correct, the automatic marker is used to check the student’s code script for equality. Equality is defined as the number of lines between the student’s and the model answer that are equal. Additionally, the student code script is normalised before checking the equality between students’ answers and the model answer. Normalisation involves the removal of redundant parts from code scripts. Hence, equality is increased between code segments. Marks are then specified for the students’ code scripts by the system if student’s and model answer are equal. If they are not equal, the student answer is marked by the human marker. Furthermore, both the system and the human marker provide comments on specific parts of code scripts such as the basic structure, input, computation and output. In this case, students receive feedback on syntax and logic errors. Manual marking also has an advantage in Sakai over automatic marking in that human markers could provide richer feedback. In addition, normalisation of code scripts can be very helpful in reducing the markers’ workload. This is because after normalisation, the number of equal code script may increase and the system can then provide feedback based on dynamic assessment using the model answers. Also, the similarity between code scripts may increase and the marker can subsequently provide
more consistent feedback based on the static assessment. However, Sakai’s approach has a few drawbacks such as:

- Equality between students’ programming codes and the model answers can generally be different. In this case, the marker must provide all possible model answers to automatically mark students’ programming codes. This could involve unnecessary repetition for the marker, since students’ solutions can vary slightly from one another, and as such, model answers need to cover each student’s solution.
- Markers might provide inconsistent feedback for similar parts of different programming codes (especially for code scripts, which do not match the model answers).
- Students do not get detailed feedback if the output is incorrect. In such cases, Sakai only provides information about the expected output as a feedback. In this sense, the feedback cannot be helpful for the students.

6. The system created by Tung et al. (2013) provides feedback for code scripts. In this system, if the student’s answer does not correctly compile and returns errors, these errors (syntax errors) are fixed by the marker. The system then runs the program again to obtain the result. This process continues until the program correctly compiles. Finally, the system provides feedback on the correction of the output utilising the dynamic assessment and logic errors. Additionally, the syntax errors are fixed by the human marker and are also reported to students as feedback. In such a manner, students can get helpful comments based on the syntax errors from the markers. Thus, they may improve their programming skills and learn how the syntax errors can be fixed. However, this marking system has some disadvantages such as:

- Human markers must improve feedback for each solution which can be a time-consuming process. In addition, human markers may also provide inconsistent feedback.
- Fixing syntax errors can be repetitive and can represent an extra workload for the human marker.

Kakkonen and Sutinen (2009) indicated that markers need to provide detailed and consistent feedback through the marking systems. In this context, the marking systems should allow markers to make detailed feedback during the marking process.
In conclusion, the potential semi-automated marking systems described above show that they do not solely focus on providing detailed and consistent (broad) feedback and improving the efficiency of the human marker in the marking systems. Furthermore, in these systems, markers provide comments based on the static assessment after the dynamic assessment for each code script in these systems. In this sense, markers’ comment(s) could be automatically repeated for similar solutions or code segments to reduce the markers’ workload. In addition, similar code pieces could be detected and displayed to the marker in the marking process. Therefore, in the context of the points highlighted above, the next section focuses on examining the clone detection techniques based on CBA.

### 2.3.2. Clone Detection in CBA

Clone detection became an active research area in software programming during the 1980s (Wani and Dang, 2015). Most of the research conducted on clone detection up until this point has focused on identical clones (Smith and Horwitz, 2009). Clone detection has been utilised for detecting plagiarism in programming, which is the main purpose of clone detection research (Smith and Horwitz, 2009). However, in this research, clone detection is part of a semi-automated assessment approach which will be used in similarity measurements between code pieces/segments based on segmented marking. Clone detection tools use similarity measurement techniques to detect similar or identical clones. As such, this section focuses on presenting the similarity measurement techniques and discusses which is the most effective. Similar code structures (segments) can be detected from different code scripts using the most effective technique. The marker’s comments can then be repeated to mark similar code segments. Students can therefore receive consistent feedback and the marker’s workload reduces.

#### 2.3.2.1. Existing Clone Detection Methods

Common clone detection methods used in the field are described in this section, which are String Matching (SM), Metric Comparison (MC) and Abstract Syntax Tree (AST).

- **String Matching (SM)** supports line-based comparisons to other lines in code scripts (Johnson, 1993). If a line is matched between code scripts, it is considered a clone (Higo et al. 2008). Roy et al. (2009) state that this method can be more effective if code scripts are normalised or transformed before the string matching. The reason is that
normalisation rules may increase the number of similar code segments among code scripts.

- **Metric Comparison (MC)** supports the calculation of several metrics. Segments are considered clones if they have similar metric values (Higo et al. 2008). This method supports comparisons of the metric values rather than comparing the code directly (Shafieian and Zou, 2012).

- **Abstract Syntax Tree (AST)** is generated from code scripts. Similar tree-parts (sub-trees) between ASTs are considered as clones (Higo et al. 2008). In this sense, a sub-tree is compared with every sub-tree of different ASTs (Rabinovich et al., 2017).

### 2.3.2.2. Effect of Clone Detection Methods in Tools

Dang and Wani, (2015) have compared clone detection tools using different clone detection methods based on parameters such as functionality, portability, scalability and accuracy. The aim of the Dang and Wani’s study was to evaluate and determine the most effective clone detection method.

**Functionality:** Tools are implemented using AST and SM methods have almost the same functionality. The tools’ functionality where AST and SM methods are used is also better than other tools implemented using the MC technique.

**Portability:** Tools implemented using AST, MC and SM methods are portable with many programming languages.

**Scalability:** The tools implemented using the SM method have high scalability. For example, Dang and Wani (2015) applied a tool implemented using the AST method, and another implemented using the SM method to a code script to detect clones. The tool based on the AST method detected only 783 clones, while the SM-based tool detected 957 clones. The clones were captured from 65 files written in Java. If a tool is able to find clones form large code scripts, its scalability is considered to be high.

**Accuracy:** If a tool finds clones accurately, it can be considered as a reliable tool (Dang and Wani, 2015). In other words, if a tool detects code segment types (e.g. if, for or loop type code segments) accurately, its reliability is considered to be high. On the other hand, if a tool can detect copied code fragments, its robustness is considered to be high. Tools implemented using the SM and AST methods are more reliable MC due to their strong ability in detecting identical and similar clones.
According to Dang’s study (Dang and Wani, 2015), the AST and SM methods are more effective clone detection methods than the MC method. Furthermore, the SM method has the highest scalability compared to the other methods. Thus, the SM method can be used effectively to find similar code segments. However, these methods have been used mostly for plagiarism detection in programming assignments/exercises and have not been commonly used for assessment. The next section considers the use of these techniques in the context of CBA systems.

2.3.3. Artificial Intelligence in CBA

Computers can be made to mimic humans’ behaviour through the use of AI techniques (Jones, 2015). AI techniques have been utilised to contribute in many areas for many years (Russell and Norvig 2016). In particular, AI techniques can make a significant contribution in CBA to provide quality feedback in education. Thus, this section provides a discussion on the use of AI in education and the reasoning systems.

2.3.3.1. AI in Education

Jones (1985) states that computer systems can be utilised for the purposes of education. Russell and Norvig (2016) highlight that systems can think deeply to solve problems, learn from experiences, find a logical way, and recognise symbols or images. Moreover, Salem (2012) highlights seven main areas of AI use in education. They are Intelligent Educational Systems (IES), Teaching Aspects, Learning Aspects, Cognitive Science, Knowledge Structure, Intelligent Tools, Shells and Interfaces. Salem (2011) states that IES is the most popular area and that it provides important support for students. As such, and since this research intends to provide supportive feedback for students (novice programmers) based on the formative assessment, only IES is discussed in this section. IES consists of two main components which are knowledge base and inference engine. Many knowledge representations and management techniques are related to the knowledge base. They refer to trees, scripts, ontologies, semantic networks and cases. The most suitable AI approach needs to be selected in order to make the systems successful in a given domain (Katoua, 2012). Inference engines include numerous approaches and methodologies. They include case-based reasoning, model-based reasoning, causal reasoning and fuzzy reasoning. Russell and Norvig (2016) highlight that the popularity of these AI approaches and methodologies in education has increased considerably, especially in recent years.
2.3.3.2. Reasoning Systems in AI

Several studies state that Case-Based Reasoning (CBR) systems have advantages compared to Rule-Based Reasoning (RBR) systems in automated supported systems (Leake, 2003). Cases are created with significant features to retrieve the case from the case-base and add it to case-base (De Mantaras, 2005). This makes CBR systems more flexible than RBR systems. Another advantage is that CBR systems can suggest solutions for similar problems (Watson, 1994). In RBR systems, rules represent knowledge using IF-THEN format. However, RBR systems have major problems including infinite chaining, addition of new and contradictory knowledge and modification of existing rules. If the system does not work properly, system developers need to add more rules. Furthermore, in RBR systems, if a new rule is used to fix a problem, it can be contradictory. Inefficiency, opacity and coverage of domain are additional disadvantages of RBR systems. Each rule is examined to understand whether it applies to the current situation during every cycle of the inference mechanism. In addition, it is difficult to examine and determine what actions are going to happen when.

Fully-automated and semi-automated assessment systems were discussed in Section 2.3.1. Most of the fully-automated systems refer to RBR systems. For instance, they focus on dynamic assessment to provide feedback on the correctness of the solution or on logic errors. If the output of the student solution and the model answer do not match, it is considered as incorrect. The reason is that execution files of these systems consist of a set of rules. In contrast to the fully-automated assessment systems, the semi-automated assessment systems should not be considered completely as RBR systems. The reason is that these systems include not only a set of rules, but also human markers to provide feedback based on the static assessment. For example, semi-automated systems mostly use normalisation rules to provide feedback and then rely on human markers to extend the feedback (Reek, 1989; Suleman, 2008). CBR has not been utilised in any semi-automated assessment systems (for computer programming) until now. This research uses CBR since it can provide solutions (which are already stored in the case-base) for similar problems (code segments).

2.3.4. Case-Based Reasoning (CBR)

Case-Based Reasoning (CBR) is an important approach in the field of education especially for problem solving and learning (Aamodt and Plaza, 1994). Generally, in the marking process, human markers provide or develop solutions according to similar situations from the previous experiences. In addition, most of the makers discuss solutions and problems based on these
past experiences (Watson, 1994). Thus, new solutions are provided based on the knowledge obtained from previous experiences.

Kolodner (2014) highlights that similar problems generally consist of similar solutions, which is the main assumption of CBR. The assumption advocates an idea which is the provision of solutions for current problems using solutions for similar past problems. In this sense, current and past problems are compared to find similarities. In this manner, the most suitable solution can be adopted for the current problem.

2.3.4.1. Motivation

Solutions are stored and then re-used for problems to enhance the problem-solving process through the use of CBR (Kolodner and Leake, 1996). CBR can be helpful in making decisions for similar problems. Furthermore, CBR is a flexible methodology compared to other reasoning methodologies such as RBR, as highlighted in Section 2.3.3. CBR enables the provision of consistent feedback since it re-uses solutions for similar problems. Furthermore, it may also reduce workload for human markers.

2.3.4.2. CBR Cycle

CBR can be described using the process cycle illustrated in Figure 2.3. The cycle has four processes which are the retrieve, reuse, revise and retain processes (Aamodt and Plaza, 1994).

Retrieval: Similar cases are retrieved from the case-base to solve the current problems. The retrieving of a given case depends on the similarity between them. Special similarity measurement techniques are applied to find a similar case, as described in Section 2.3.2. If problems are similar, the case is retrieved by the system.

Reuse: The solution of the retrieved case can be adopted to the current problem directly. The current problem is then marked accordingly.

Revise: The adopted solution is revised to provide the best solution for the current problem if necessary. The adopted solution cannot be confirmed directly to the current problem without revision.

Retain: The current problem and adopted or revised solution are considered as a new case. The new case is stored with other cases in the case-base. The new case can be also retrieved to solve new problems in the future.
Paper-based assessment and semi-automated assessment systems have the same issue in that they may provide inconsistent feedback for similar solutions. The CBR cycle provides an opportunity to provide efficient and consistent feedback for similar problems based on the dynamic assessment. Furthermore, the CBR cycle may also contribute towards reducing markers’ workload.

2.4. Conclusion

This chapter described the importance of the formative assessment and semi-automated assessment approaches based on CBA. The semi-automated assessment approach is more beneficial in providing personalised feedback based on the formative assessment for novice programmers than the full-automated assessment approach. However, current semi-automated assessment systems are not effective in terms of providing consistent feedback. The reason is that human markers may provide inconsistent feedback for similar code scripts based on static assessment when the number of students in a classroom is high. However, if human markers’ comments are re-used for similar solutions based on static assessment, students can naturally obtain consistent feedback. Thus, a new marking technique is required in order to provide consistent feedback in a more effective manner than current marking systems based on semi-automation. Additionally, CBR cycle needs to be adopted in the new marking technique since reusing previous solutions for similar problems may contribute towards providing consistent feedback.
A new semi-automated marking approach is developed in Chapter 4 to overcome the current problems and challenges. The new approach introduces a novel-marking model based on CBA, and the CBR cycle is adopted in the marking model to provide efficient, personalised and consistent feedback.
CHAPTER 3
Research Methodology

3.1. Introduction

Research methodology is an important part of answering research questions (Bryman, 2015). It specifies the used methods in research studies and describes how a study is carried out (Vaishnavi and Kuechler, 2015). The overall research process involves the selection of techniques used to gather and analyse data. This chapter presents different research philosophies, approaches, methods and strategies that could be used in order to provide solutions for research questions. In addition, the advantages and disadvantages of these approaches and methods are briefly examined in order to choose the optimum methodology for this research. This chapter also provides information on the time horizons and validity and reliability of this research study.

3.2. Research Onion

Two models are commonly used to provide a viewpoint of the research methodology, namely the nested method and the research onion (Bryman, 2015). Despite the fact that both models consist of steps that need to be followed for an effective research, the research onion is a more detailed model compared to the nested model. The research onion was developed by Saunders at al. (2012) and depicts different research stages. Figure 3.1. illustrates the research onion consisting of different layers. The outer layers refer to the research philosophies and approaches that may be adopted in a research. The middle layers refer to the methodological choice, research strategy(ies) and time horizon. They contribute to underpin the overall research design. Finally, the core layer is concerned with data collection and analysis.
The research onion model was adopted for this research since it provides a broad perspective as a descriptive model. The rest of this chapter defines the various stages in the research process and presents the reasons for the choices made in conducting this research study.

3.3. Research Philosophy

A research philosophy refers to the set of beliefs concerning the nature of the reality being investigated (Bryman, 2012). The research philosophy is a way of deciding which data needs to be gathered, analysed and used when investigating a particular phenomenon (Saunders at al. 2012). Myers (1997) categorises four research philosophies: positivism, realism, interpretivism and pragmatism. The preferred research philosophy can contribute towards explaining the assumptions inherent in the research process. This section defines and describes the four research philosophies in the context of this thesis.

3.3.1. Positivism

All information should be factual knowledge obtained through observation (Myers 1997). Researchers highlight that phenomena should be isolated, and not dependent on study (Levin
1988). Positivism focuses on causality, and as such, simple elements are reduced in phenomena (Myers and Avison, 2002). In addition, programming and logical means are used in order to generate and test hypotheses. Experiments, case studies and surveys can be utilised to gather data. In positivist studies, researchers are restricted in terms of data collection and interpretation (Saunders et al., 2012). Since this research’s findings are mostly observable and quantifiable, this research philosophy can be considered an option based on the purpose of the research questions.

3.3.2. Realism

Saunders at al. (2012) state that realism is similar to positivism since it focuses on observable data and facts. In addition, a research does not provide biased results because realism depends on the idea of independence from the human mind. According to realism, scientific approaches/methods are not perfect. In addition, realism refers to the belief that all theory can be revised and cannot be definitively proven without continually researching (Vaishnavi and Kuechler, 2015). Inaccuracies in sensations may arise if insufficient data exist while forming realism-based research. This is known as direct realism. Alternatively, sensations may be open to misinterpretation, which is known as critical realism (Saunders et al., 2012). As such, this research philosophy cannot be an appropriate option for this research since inaccuracies in sensations may lead to incorrect results.

3.3.3. Interpretivism

Interpretivism opposes positivism since it integrates human interests into the study. According to interpretivism, researchers may gain an understanding based on social roles when they interpret data (Bryman, 2015). Additionally, interpretive researchers start out with the assumption that only social constructions such as such as language and consciousness provide access to reality (Collins, 2010). Interpretive research does not redefine variables (dependent & independent), and instead focuses on the complexity of the human sense (Runeson and Höst, 2009).

3.3.4. Pragmatism

In pragmatic studies, observable phenomena can satisfactorily provide knowledge based on the research questions (Collis and Hussey, 2014). Thus, different perspectives are integrated by practical and applied research, which contribute to interpreting the data (Saunders et al., 2012).
Additionally, the pragmatic philosophy enables researchers to use any suitable method for the research problem in a study (Collis and Hussey, 2014). In this sense, researchers are not restricted in terms of data collection, which may lead to providing biased results.

3.3.5. This Thesis’ Research Philosophy

Despite researchers often advocating that one philosophy is better than others, Podsakoff et al. (2012) states that one philosophy is not inherently better than others. Research relating to science mostly prefers the positivism approach. This is because factual data is obtained through observation. Furthermore, researchers are restricted in terms of data collection by the objective approach, and the research findings are mostly observable and quantifiable. Since this study presents a new semi-automated assessment tool based on CBA, the positivism approach allows obtaining factual data and observations from using this tool and testing it.

3.4. Research Approach

In research studies, research approaches are helpful in determining which approach is suitable for the research study (Bryman and Bell, 2012). The research approach can be considered as a type of reasoning adopted in a research or study. According to Saunders et al. (2012), there are three main research approaches, namely deduction, induction and abduction.

3.4.1. Deductive

In the deductive approach, a specific hypothesis is developed based on a pre-existing theory, and the validity of the hypothesis is subsequently tested (Pelissier, 2008). The logic of deductive inference highlights that conclusions must be same as the premises. In addition, generalisation is from the general to the specific (Wilson, 2014). Data is collected to assess a specific hypothesis relating to a pre-existing theory. Hypotheses can be formulated and results can be analysed using statistical tests in the deductive approach. Thus, the deductive approach is considered generally related to the positivist philosophy (Gulati, 2009; Snieder and Larner, 2009).

3.4.2. Inductive

In the inductive approach, generalisation is from the specific to the general, unlike the deductive approach (Bernard, 2011). This approach supports multiple methods ranging from specific observations to theories (Saunders et al. 2012). Additionally, in this approach,
observations act as the starting point for researchers. The logic of inductive inference highlights that untested conclusions are generated using known premises. Data is collected to explore a phenomenon, identify a theme and create a conceptual framework (Lodico et al. 2010).

3.4.3. Abduction

The abduction approach focuses on addressing weaknesses related to the deductive and inductive approaches (Saunders et al. 2012). The logic of abduction inference state that testable conclusions are generated using known premises. Furthermore, generalisations are established from the interactions between the specific and general. Data is collected to explore a phenomenon, identify a theme and create a conceptual framework as in the inductive approach.

3.4.4. This Thesis’ Research Approach

The abduction approach will be used in this thesis since it involves both the deductive and inductive approaches’ reasoning. The flexibility provided through both approaches may enable the generation of better ideas (Gregory and Muntermann, 2011). The presented research questions in this study involve the design, development and testing of marking tools. Through the abduction approach, a conceptual model will initially be developed in this research to form the base for data collection and analysis. Both the deductive and inductive approaches will be utilised for the development process. A series of hypotheses will be established using the model, which will subsequently be tested in this research.

3.5. Research Methods

The layer of the research onion relating to the research methodology (see Fig. 3.1.) involves the mono and multiple methods. The mono method involves using one research method such as quantitative and qualitative. In addition, the multiple methods require more than one research method. This layer highlights the basic but essential choices that researchers face when designing their research (Saunders et al., 2012).

3.5.1. Quantitative Research Design

The quantitative method is mostly related to the deductive approach (Bryman, 2015). Hypotheses and theory are tested by the researcher using the data. In addition, this method can involve an inductive approach, where data is used in order to develop a theory. Furthermore, numerical relationships are examined in quantitative studies. The numerical relationships are
then analysed through statistical techniques in order to control and ensure the validity of data. The quantitative method can provide more effective results after using statistical techniques when a study consists of a large number of participants (Saunders et al., 2012).

### 3.5.2. Qualitative Research Design

The qualitative method relies on an interpretive philosophy and measures subjective data as opinions and feelings rather than numerical data (Denzin and Lincoln, 2011). Researchers collect data to generate new hypotheses and theories during filed work. The qualitative method mostly relates to the inductive approach, where the research design is used in order to improve valuable theoretical perspectives. In some cases, this may be incorporated with the deductive and abduction approaches (Saunders et al., 2012).

### 3.5.3. Multiple Research Design

Multiple methods refer to the use of both quantitative and qualitative research methods in a research study (Creswell, 2013). Oliver-Hoyo and Allen (2006) state that both the quantitative and qualitative methods can be combined in this method by researchers. Furthermore, collecting and reviewing the data through the different methods should be conducted carefully in order to achieve a more accurate and valid estimate of a construct. The advantage of the multiple method approach is that researchers can divide the research into separate parts. The techniques obtained from quantitative methodology and qualitative methodology can then be used to analyse each part (Jackson, 2011).

### 3.5.4. This Thesis’ Research Method

The multiple method will be used in this research. This is because the multiple method utilises the advantages of both the quantitative and qualitative methods, which gives the researcher a chance to examine the research problem through a variety of ways. Furthermore, interpretation is continual and influences all stages in the research process. This enables the provision of detailed information on the examined study.

### 3.6. Research Strategies

The layer of the research onion about the research strategies shows how a researcher prefers to carry out his/her research (Saunders et al., 2012). For instance, if the inductive research approach is selected for a research, the preferred strategies are often more experimental. On
the other hand, the preferred strategies can be active or ethnographic if the deductive research approach is selected. Numerous research strategies have been proposed, with the most common research strategies presented in this section.

1. Experimental Research

All factors which may affect the result of a study are controlled by researchers. As such, researchers often try to predict what may occur (Salkind 2010). The results of an experiment are compared with the expected results. Experimental research can be used in all areas of research and it investigates the relation between factors in order to evaluate the research outcomes.

2. Surveys/Questionnaires

Surveys/Questionnaires consist of questions used to gather information from participants about their opinions and behaviours (Umbach 2004). They are mostly used in quantitative studies (Saunders et al., 2012). In addition, a survey allows the collection of large amount of information (quantitative data) in a short time period (Kitchenham 2006). The quantitative data can then be empirically analysed, where it is mostly used in order to investigate a causal relationship between different types of data.

3. Interviews

It is a flexible research strategy since it allows the participant to interpret the questions according to his/her own perspective (Bryman, 2015). In a research, results on quantitative data can be unintended in the context of the research questions. This could be often attributed to the fact that participants can be sometimes restricted in the survey or questionnaire since they are required to choose one or two answers from an answer list. However, participants may provide different results to those of the quantitative data in an interview. This can thus lead to providing valuable contributions towards answering the research problems.

4. Case-Study Research

The case-study research strategy refers to an in-depth examination of a particular problem (Oliver 2010). This strategy investigates the answers to particular problems. Individual opinions are examined in this strategy which makes it rather useful for studies. In addition, the strategy can reveal the key features of different cases (Silverman, 2013). Case-studies provide qualitative data (Yin 2003; Flick 2015) and are mostly used in social science
studies. They aim to find answers of ‘how’, ‘why’ and ‘what’ in issues under research examination.

5. Action Research

Action research follows a characteristic cycle where problem understanding is developed and plans are made for an interventional strategy. In different forms, suitable observations are collected to fulfil this intervention. This cycle is continued until the problem is sufficiently understood through a new interventional strategy (Coghlan et al. 2014). Through detailed observations, highly reliable data can be gathered. However, the disadvantage of action research is that the cause, effect and outcome lack generalizability since they may yield different results.

6. Ethnography

Ethnography focuses on observing people in order to analyse their intentions and cultural interactions (Saunders, 2011). Furthermore, ethnography is used to interpret the link between human culture and behaviour. In ethnographical research processes, researchers aim to conduct the research using people’s perspectives through observation. However, in this research strategy, complex behaviours can be described and causal conclusions are impossible.

3.6.1. This Thesis’ Research Strategies

Three research strategies have been chosen for this research study, namely experimental research, surveys/questionnaires and interview. This study investigates the provision of feedback via a new marking technique for code scripts written by novice programmers. An experimental study will be carried out to measure the feasibility of the new marking technique. Subsequently, a survey/questionnaire and an interview will be conducted to capture the participants’ thoughts on the new marking technique. In addition, the marking tool will be designed and implemented based on the results of the feasibility study. Furthermore, the marking tool will also be evaluated based on the experimental study to compare their efficiency based on the quantitative measurement results.
3.6.2. Experiments and Sampling

Experimental research will be utilised in this study, as mentioned above. The experimental research will support the examination of the feasibility of the new marking technique and the usability of the developed marking tools. In experimental research, the sampling methods used are simple random sampling, systematic sampling, stratified sampling, cluster sampling and purposive sampling (Brown, 2006; Devore, 2015).

- In simple random sampling, each member is selected randomly and each participant has equal chance to be selected. Large sample size (a few hundred) is used in this method and therefore, application of it in practice is not be easy (Gravetter and Forzano, 2011; Saunders et al., 2012). If participants have the same experience on a subject or a topic, this sampling method does not pose any issues for the research. However, it is not easy to find a few hundred participants who have same experience in large sample size experiments. Thus, this sampling method cannot be effective for this research.

- In systematic sampling, every Nth member is used which can be effective in large sample sizes (Bajpai, 2010). However, this research sample size is small and as such, the use of this sampling method is not effective for this research.

- In stratified sampling, members are divided into two or more subgroups called “strata”. Members are selected from these strata using either random or systematic sampling methods (Proctor, 2003). This sampling method can provide unreliable results since members are selected from the groups may not have the same level of knowledge on the subject.

- In cluster sampling, members are divided into groups referred to as “blocks” or “clusters” (Brown, 2006). Clusters are chosen randomly and each member in these clusters is used. This sampling method can pose an issue if the members in a cluster do not have the same experience on the subject of the study, which could lead to producing incoherent results.

- In purposive sampling, members are chosen by the researcher. Researchers believe that the chosen members are representative samples and their personal judgements help answer the research/study questions (Saunders et al., 2012). Furthermore, small sample sizes can be used in this sampling method (Guarte and Barrios, 2006).

Because of the time limitation on completing the study, the purposive sampling method will be used in this research. In the purposive sampling method, a small sample size can be used. In
this sense, the developed marking tools can be tested by expert markers who have experience on recent and relevant teaching techniques of code scripts in order to obtain reliable and accurate results. Bloxham (2009) and Paré and Joordens (2008) state that markers should have many years of experience in providing feedback to be qualified as experts. However, they did not specify exactly how many years are required to be an expert marker. In this study, a marker with at least ten years of experience is considered as an expert marker. This is due to the fact that a marker who possesses a marking experience of ten years or more is able to provide valid and reliable data. The purposive sampling method will thus be used to evaluate the developed marking tool. However, finding expert markers is difficult since many have busy schedules and are thus not keen on participating in the testing of marking tools. Hence, the sample size chosen in this study will inevitably be small. Holzinger (2005) state that a small sample size (5-20) is sufficient and Creswell (1998) highlights that researchers can collect valuable results using at least 5 participants in a study. In this research, the marking tool will be tested using 8 participants, five of whom will be expert markers, so that reliable results can be obtained. In addition, two experiments, namely a pilot and a main experiment, studying the feasibility of the marking technique will be carried out before the marking tool evaluation. The main experiment will be conducted with 12 participants. Connelly (2008) state that the sample size in a pilot study/experiment should be at least 10% of the main experiment. As such, four participants are considered sufficient for the pilot study.

3.7. Time Horizons

The research onion highlights two types of time horizons, which are the cross-sectional and the longitudinal time horizons (Figure 3.1) (Bryman and Bell, 2015). In cross-sectional time horizon, data must be collected in a short period of time (Flick, 2015). In contrast, data is collected in an extended period of time in the longitudinal time horizon (Bryman and Bell, 2012). Mostly, the selected time horizon is not dependent on a specific research methodology (Saunders et al., 2012). This research adopts the cross-sectional time horizon in order to collect and evaluate data that supports the development of more efficient computer-based assessment tools based on the timeframe of this study.

3.8. Validity and Reliability

Research questions will be answered by collecting quantitative and qualitative data, as which was discussed in Section 3.4. According to Oliver (2010), taking steps is a crucial process for
ensuring the fulfilment of the basic requirements in terms of validity and reliability of a study. The rest of this section describes this process in the context this research.

### 3.8.1. Validity

Validity covers the experiment’s conception and finds whether the measurement results are expected according to the requirements of research methods (Wilson, 2014). To ensure the validity of the measurements in this study, the following steps will be taken:

- Experiments will be designed according to the steps outlined in the research methods.
- The pilot experiment will be carried out before the main experiment to analyse the participants’ work and gather thoughts on the proposed marking techniques.

### 3.8.2. Reliability

Reliability means that any significant results must be inherently repeatable (Cohen, 2013). To ensure the reliability of the measurements in this study, the following steps will be taken:

- The user interface of the marking tool will be designed based on the findings from the feasibility study.
- Participants will be observed in the experiments to interpret the experimental results in a more detailed fashion.
- Participants will answer a questionnaire after the experiments to obtain factual data and observations.

However, this research cannot generalise the results with a high degree of reliability because of the small sample size. Nevertheless, even with the small sample size used in this research, expert markers/participants can provide reliable, worthwhile and sufficient results in order to justify the research questions. Thus, the use of expert markers is important factor in terms of enhancing the reliability of the results and findings of this research.

### 3.9. Research Process

Takeda et al. developed a design cycle of the research process (Takeda et al. 1990). Vaishnavi and Kuechler applied Takeda’s cycle to their own study (Vaishnavi and Kuechler 2015). Table 3.1 illustrates the research design framework developed by Takeda et al.
Table 3.1 Research Design Framework (modified by Vaishnavi and Kuechler 2015)

<table>
<thead>
<tr>
<th>Process Steps</th>
<th>Outputs</th>
<th>Knowledge Flows</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Awareness of Problem</td>
<td>Proposal: Identification &amp; definition of the problem and its context.</td>
<td></td>
</tr>
<tr>
<td>2 Suggestion</td>
<td>Tentative Design: Preliminary solution to the problem.</td>
<td></td>
</tr>
<tr>
<td>3 Development</td>
<td>Artefact: Solution developed with many iterations and feedback.</td>
<td>Circumscription ↑</td>
</tr>
<tr>
<td>4 Evaluation</td>
<td>Measures: Evaluation of solution against original specifications.</td>
<td></td>
</tr>
<tr>
<td>5 Conclusions</td>
<td>Results: Conclusion and communication of research effort.</td>
<td>Operational &amp; Goal Knowledge ↑</td>
</tr>
</tbody>
</table>

The research design framework in Table 3.1 consists of different process steps. Since any research can be designed based on this framework, these steps are mapped to this thesis’ structure as illustrated in Figure 3.2.
In conclusion, this research’s framework consists of all process steps in Table 3.1, and its research is based on the positivism philosophy.

### 3.10. Conclusion

This chapter provided detailed information on the research philosophy, method and strategies used in this research study based on the research onion model. Initially, various approaches and methods were briefly described and discussed, after which, the appropriate choices for this research were presented. This research intends to obtain factual information through
observation, which motivated the selection of the positivist philosophy for this research. In addition, numerical relationships are required to ensure the validity of data in this research (quantitative method). Furthermore, subjective data (e.g. opinions) needs to be interpreted in order to provide valuable results based on the qualitative method. The use of both methods (i.e. the multiple research method) in this research thus enables the collection of more reliable data, therefore strengthening the research findings.

This research advocates a novel way of marking, and marking models are developed based on this new way of marking. Thus, experiments are required in order to measure the feasibility of this novel marking method and evaluate the marking models. As such, experimental research was chosen as a research strategy for this study. In addition, since surveys/questionnaires and interview are a sensible way to capture participants’ thoughts about the proposed novel way of marking in a short time period, they were also chosen and will be carried out after each experiment. Building on these choices, the subsequent chapters will present the theoretical and practical details of the study.
CHAPTER 4
A New Semi-Automatic Assessment Framework for Programming Code

4.1. Introduction

CBA systems have been discussed in Chapter 2 based on their marking approaches. These systems provide feedback on the general structure of code scripts. The importance of semi-automated assessment systems to formative assessment has also been highlighted in Chapter 2. In addition, paper-based assessment approaches (based on the program code only) have been examined in order to computerise the marking process as much as possible. Chapter 2 has also identified several issues with paper-based assessment approaches, such as providing consistent feedback and reducing markers’ workload. This chapter proposes a novel semi-automated assessment approach which is a hybrid of the full and paper-based assessment approaches. It aims to help markers in the marking process provide consistent and efficient feedback and reduce their workload.

This chapter also briefly discusses the challenges of the manual (paper-based) assessment process (Section 4.2). Section 4.3 discusses an observation made by this research’s author related to novice students’ programming solutions. The subsequent section then presents the rationale for using the proposed semi-automated assessment approach. The approach is defined as a broad overview and outlined in Section 4.5. Finally, the marking process is described in detail in Section 4.6.

4.2. Manual Assessment Process Cycle

The manual (paper-based) assessment process cycle, described in Section 2.2.2.1, consists of four generic processes: question preparation, obtaining program code, marking and grading, and feedback generation. Each process is associated with a particular challenge, with the most important challenge being to provide consistent feedback in the marking process. This stems from the fact that human markers may lack consistency when marking students’ programming codes. In addition, the provision of feedback for general structures instead of specific (code
segments) parts of a code script poses another challenge. This is because providing feedback for each code segment in a code script can be a time-consuming process for markers. Therefore, many researchers have studied and investigated CBA in the context of providing consistent and efficient feedback. This research also aims to provide consistent feedback as well as reduce markers’ workload through the semi-automated assessment approach.

4.3. Observation

This section analyses different aspects of typical source code of novice programmers. It aims to comment on the repetitive elements of students’ program code. All programming languages could potentially be assessed utilising the repetitive elements of the program codes. That is, this approach can be applied equally well independent of the programming language. Previous observations by this project’s researcher indicate that students’ code structures generally contain similar code segments. Such code segments often contain ‘if’ or ‘loop’ types control statements. More detailed information on such code segments is provided in Section 4.6.1. Table 4.1 gives an example on the similar code segments among students’ code scripts.

<table>
<thead>
<tr>
<th>Table 4.1 Code Scripts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Script 1</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

Each of the letters in Table 4.1 refers to a common code segment. As can be seen from Table 4.1, code scripts typically consist of repetitive code segments. For instance, each code script in the example above consists of code segment A. In the marking process, the marker can choose to provide feedback for only one code segment out of all repetitive code segments. For instance, code segment A under Code Script 1 could be marked by a human marker. Then, the marker’s comments can be copied to mark the rest of the repetitive code segments based on the CBR cycle introduced in Chapter 2. In such a manner, the rest of the repetitive code segments can be marked automatically. For instances, code segments A in Code Script 2, Code Script 3 and Code Script 4 can be automatically marked. Furthermore, the similarities between code segments can be increased to obtain more repetitive code segments using a set of rules.
before the marking process. The marker’s comments can therefore be applied to more repetitive code segments which enhances the feedback consistency.

4.4. Rationale for the Semi-Automated Assessment Approach

Section 2.2.2 provided information on assessment approaches such as paper-based, full-automated and semi-automated approaches. A paper-based assessment approach is very helpful in providing personalised feedback. However, the marker generally cannot provide consistent feedback because paper-based assessment tends to be time consuming (Ihantola et al., 2010). Assessment systems based on the fully-automated and semi-automated assessment approaches were discussed in terms of providing consistent and personalised feedback in Section 2.3.1. The key advantage of fully-automated assessment systems is their ability in providing consistent and instant feedback. These systems are used extensively for the assessment of objective questions (e.g. multiple choices questions and true-false questions), as this type of questions requires one correct answer. However, full-automated assessment systems are not commonly used for free response questions, where students typically provide different solutions to the same question. In this case, the full-automated assessment systems may not provide effective feedback for students. It can be inferred that human markers can be still considered as a key factor for free response type questions in CBA systems due to the following reasons:

1. Students may receive more detailed and personalised feedback from a human marker.
2. The marker may provide more feedback on certain parts of code script.
3. The marker may provide detailed feedback if a student’s solution is incorrect or incomplete.

However, human marker does not provide feedback directly when using a fully-automated assessment system. Yang et al. (2002) state that full-automated assessment systems may not provide sufficient feedback on the content of students’ solutions. Furthermore, Dikli (2006) highlighted that if students know the feedback is generated automatically, their motivation may drop. Thus, free-response type questions should not be assessed using full-automated assessment approaches (Suleman, 2008).

In semi-automated assessment systems, students receive more detailed and personalised feedback on the content of their solutions. The semi-automated assessment systems allow human markers to provide feedback while the computer supports the saving and re-use of the
marker’s feedback for similar students’ solutions. A semi-automated assessment system is therefore capable of providing more efficient feedback using the advantages of both the paper-based and the full-automated assessment approaches.

4.5. Overview of the proposed Semi-Automated Assessment Approach

Figure 4.1 illustrates the proposed semi-automatic assessment approach cycle used in this research. The proposed assessment approach consists of seven processes. The rest of this section describes each of the processes. The marking process is indicated by the blue rectangle (core process) in Figure 4.1.

- In Process 1, the marker prepares a question which highlights the important parts of the subject such as loop type, print messages etc. If the marker highlights the important parts in the questions, students provide appropriate solutions accordingly.

- In Process 2, students produce their programming code according to the question requirements. Students could use a generic IDE to provide well-formed solutions. Students then submit their programming code using online, rather than offline submission, since offline submission can be an extra workload for markers.

The core process in Figure 4.1 consists of Process 3, Process 4, Process 5 and Process 6.

- Process 3 generates code segments from students’ code scripts. CBA systems do not yet provide feedback on code segments in a code script at this point. They generally provide feedback on the structure of a code script. Code scripts generally consist of similar code segments if question requirements are highlighted well, as described in Section 4.3. For example, if the similarity between two code segments is more than 90%, the code segments can be considered similar or repeated even if they are not identical. In this sense, obtaining code segments from different code scripts increases the number of similar code segments. In addition, if a marker provides feedback on code segments in a code script, students can benefit by receiving more detailed and personalised feedback. Since this research aims to provide segmented feedback as described in Section 4.1., Process 3 is utilised for parsing the code scripts to obtain code segments.
• **Process 4** normalises each code segment by applying generic rules to increase the similarity between code segments. This results in increasing the repetitive parts in the code scripts, which affects group numbers in Process 5.

• **Process 5** groups similar components into a single group. It initially applies a similarity measurement technique to find similar components. Then, a threshold is applied whereby if any two components’ similarity exceeds the threshold, they are assumed to be similar and put into the same group. Further detailed information about setting threshold is presented in Section 4.6.3. Since similar components receive the same comments, this process effectively increases the feedback consistency.

• Process 6 is the marking process. The marker provides comments for only one code segment from each group. Then, the process uses the marker’s comments to automatically mark the rest of the code segments. This results in enhancing the feedback consistency and reducing the marker’s workload.

• In Process 7, feedback is generated by the semi-automated assessment approach. The feedback mentions the common mistakes that students’ code scripts contains, as discussed in Section 2.2.3. Each student also receives personalised feedback.

![Fig. 4.1 The proposed semi-automated assessment approach cycle](image-url)

The core process of the proposed semi-automatic assessment approach is very important since it advocates a new way of marking to provide more detailed, personalised and consistent
feedback and reduce markers’ workload. The rest of this chapter examines Process 3, Process 4, Process 5 and Process 6 in more depth.

4.6. Core Process of the Approach

The semi-automated assessment approach adopted in this research faces some challenges related to the marking of code scripts, which can be summarised as follows:

1. How code segments can be obtained from code scripts?
2. How similar (repetitive) code segments can be obtained from different code scripts?
3. How the marker’s workload can be reduced?
4. How consistent feedback can be provided based on the semi-automated assessment approach?

This section describes the core process of the semi-automated assessment approach and investigates how to address the questions described above.

4.6.1. Segmentation Process

CBA systems generally provide feedback on the correctness and the general structure of the submitted solutions. However, they do not provide feedback on each code segment in a code script. Providing feedback on each code segment in a code script is referred to as segmented marking in this research, which is a novel way of marking. As discussed in Section 4.3., code scripts generally consist of similar code segments. If code scripts are parsed to obtain code segments, the number of similar code segments can increase. The parsing criteria of code scripts are presented in the next section. This section describes the code segment types and discusses the code segment type hierarchy. Code segments have three types: sequential, if statements and loop (Liang, 2013). Liang (2013) describes the code segment types as basic statements in his research, where his objective was to introduce basic statements to novice programmers. In addition, these code segment types are prevalent in different programming languages. A code segment type hierarchy is illustrated in Figure 4.2, where the rectangles refer to code segments and the code segment types are shown in ellipses. As can be seen from the figure, code segments consist of three types. The dashed line in Figure 4.2 means that the code segment type hierarchy may consist of more code segment types. In this research, a sequential type code segment means that the code segment does not contain a control statement.
Furthermore, “if type” code segments include three sub-types, which are “simple if type”, “else type” and “else-if type” statements. Similarly, the “loop type” code segments refer to three different sub-types, which are “for loop type”, “while loop type” and “do-while loop type”.

![Code Segment Type Hierarchy](image)

**Fig. 4.2 Code Segment Type Hierarchy**

Furthermore, each code segment type (except the sequential type) consists of control, condition and code segment structures. They are not illustrated in Figure 4.2 since these structures will be used in Process 5 (the grouping process) to measure the similarity between code segments. Table 4.2 shows an example of code segments within a code script that is five lines long. For example, code segment 1 is a “sequential type” code segment, whereas code segment 2 is an “if type” code segment.

**Table 4.2 Code Scripts**

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Programming Code</th>
<th>Code Segments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><code>import math</code></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td><code>r = int(input(&quot;enter a value for the r:&quot;))</code></td>
<td>Code Segment 1</td>
</tr>
<tr>
<td>3</td>
<td><code>h = int(input(&quot;enter a value for the h:&quot;))</code></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td><code>if h &lt; 0 and r &lt; 0:</code></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td><code>print 'r and h are negative'</code></td>
<td></td>
</tr>
</tbody>
</table>
In Table 4.2, students may get more detailed feedback if the marker provides feedback on code segment 1 and code segment 2 separately. In this case, the student benefits more from receiving separate comments on each code segment structure than receiving general comments on their programming code. Therefore, code segments need to be parsed based on the code segment types.

### 4.6.1.1. Parsing Criteria

Parsing criteria are important for identifying code segment types. In this research, segmented marking supports commenting on each code segment in a code script. Therefore, code segments need to be highlighted to the marker in the marking process, who can then provide comments for each code segment based on segmented marking. Code segments are displayed with the following notation in this process: each code segment has a start line and an end line, which is a standard format. However, some code segments utilise nested code segments. In this case, the nested code segments also need to be commented on based on the segmented marking process. Thus, the following notation is created based on segmented marking.

\[
[CStype(SL,EL)[CStype(....)]\]

Where

- **CStype** = Code Segment Type
- **SL** = Start Line
- **EL** = End Line

The square bracket indicates the nested code segments in the proposed notation. Code segment 1 and code segment 2, in Table 4.2, can be written based on the code segment notation as follows: \([\text{SEQUENTIAL}type(1,3)]\). According to this notation, code segment 1 is a sequential type code segment; it starts at the first line and ends at the third line. Moreover, code segment 2 is expressed as \([\text{IF}type(4,5)]\). Similarly, according to this notation, code segment 2 is an if type code segment and it starts at the fourth line and ends at the fifth (last) line.

At the end of the segmentation process, code scripts are parsed and code segments are subsequently obtained. As discussed after Figure 4.2, each code segment type except the sequential type consists of control, condition and code segment structures. These structures are used to measure the similarity between code segments in the grouping process. Thus, code segments need to also be parsed to obtain the control, condition and code segment structures of the code segments. The rest of this process describes the parsing of code segments. It presents and describes the control, condition and code segment structure trees.
- **Control Structure Tree**

The aim of this tree is to obtain the control structure of code segments. Figure 4.3 illustrates a control structure tree using basic shapes. The rectangle refers to a control structure and the ellipses refer to the types of control structure. According to Figure 4.3, the control structure of a code segment may refer to different types of control statements, such as “if” or “for” type statements. The dashed line in the figure means that the control structure may be a different type of control statement, such as “while”, “if-else”, etc.

![Control Structure Tree]

**Fig. 4.3 Control structure tree**

The code segment is parsed to obtain its control statement. For instance, if any line starts with a control statement (if, for, while, etc.), the statement is parsed and it is accepted as a code segment structure.

- **Condition Structure Tree**

Figure 4.4 illustrates the condition structure tree using basic shapes. The rectangle refers to the condition structure and the ellipses refer to the type of the condition structure. According to Figure 4.4, a condition structure may consist of more than one condition. The dashed line in the figure indicates that a condition structure may consist of other conditions or of different combinations of conditions and logic operators. However, since this research aims to provide feedback for novice programmers, it only utilises two condition structures as illustrated in Figure 4.4. However, this study can also support different combinations of conditions and logic operators if its scope is widened in the future to include the provision of feedback for expert...
programmers. For instance, a condition structure may consist of more than two conditions, and more than one type of logic operators can be also used in a condition structure.

![Condition Structure Diagram](image)

**Fig. 4.4 Condition structure tree**

Code segments are parsed to obtain their condition structures. If any line starts with a control or loop statement (if, else, for, while etc.), the line is parsed without the control or loop statement and is accepted as a condition structure.

- **Code Segment Structure Tree**

The block part of a code segment (except the control and condition parts of a code segment) can be equal to any type of code segment.

### 4.6.2. The Codifying Process

Process 4 consists of generic rules which are used to normalise code segments. Code segments are transformed into components at the end of this process, as illustrated in Figure 4.2. The importance of Process 4 in this research is that it increases the similarity between components through the application of generic rules. Process 5 initially measures the similarity between various components and then groups them based on the similarity measurement ratios. The string matching technique is used in the similarity measurement process. The reason behind choosing this technique was already described in Section 2.3.2. In the marking process, human markers provide feedback for a group member only. The rest of the group members are subsequently marked automatically. A human marker’s workload may reduce if the number of group members increases. In this case, before the grouping process, the codifying process applies generic rules to increase the number of group members. The codifying process directly affects the grouping and marking processes in terms of providing
consistent feedback. In addition, the normalised code (component) is not seen by the marker in the marking process. The normalised component is merely used to group components appropriately.

Process 4 applies generic rules to all code segments. The rules were created in this project based on the python programming language, as many experts consider Python to be the best language for novice programmers (Hetland 2006). However, the rules could be also extended to other programming languages if required, such as C, C++ and Java. The rest of this section presents and discusses the generic rules.

**Blank Line Rule:** This rule is applicable to any code segment types. If there are blank lines in a code segment, they are removed for the purposes of similarity measurement, since blank lines do not add semantic meaning. Otherwise, the presence of blank lines could result in components not matching even though they are effectively the same. Table 4.3 shows an example of the blank line rule.

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Before applying the rule</th>
<th>After applying the rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a = y + x</td>
<td>a = x + y</td>
</tr>
<tr>
<td>2</td>
<td>blank line</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>print a</td>
<td>print a</td>
</tr>
</tbody>
</table>

Before applying the rule, line 2 is identified as a blank line and subsequently removed based on the requirement of the blank line rule.

**Comment Rule:** This rule is also applicable to all code segments. If there are any comments or comment lines in the code segment, they are removed to increase the similarity between components. Students mostly provide comments to explain their code. However, identical comments (in terms of meaning) cannot be matched due to their likely differing syntax. Therefore, comments and comment lines are removed by applying this rule. However, markers provide feedback on code segments rather than components in the marking process. As such, markers are still able to see the comments in the marking process. Table 4.4 displays an example of the comment rule.
### Table 4.4 Example of the comment rule

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Before applying the rule</th>
<th>After applying the rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a = x + y #comment</td>
<td>a = x + y</td>
</tr>
<tr>
<td>2</td>
<td># comment line</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>print a</td>
<td>print a</td>
</tr>
</tbody>
</table>

In the example, before the application of the rule, Line 1 consists of a comment and line 2 is comment line. The comments are removed based on the comment rule requirements, as illustrated in Table 4.4.

**Parentheses Rule:** This rule is applicable to any code segments consisting of one or two conditions. If any condition is written in parentheses, the system removes the parentheses. For example, a condition is written in parentheses while another is written without parentheses. In this case, they do not match even if they are the same semantically. Table 4.5 displays an example of the parenthesis rule.

### Table 4.5 Example of the parentheses rule

<table>
<thead>
<tr>
<th>Before applying the rule</th>
<th>After applying the rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x &lt; 3)</td>
<td>x &lt; 3</td>
</tr>
</tbody>
</table>

In addition, a condition can be written using more than one set of parentheses such as ((x<3)). In this sense, removing the parentheses can be more helpful than adding them based on the aim of the codifying process. Furthermore, removing the parentheses does not affect the semantic of the code script, since this rule is applicable to short and simple conditions consisting of a maximum of two conditions (e.g. “(x<3) and (y<5)”). It worth mentioning that if more complex conditions were used in this research (e.g. “(x<3) and ((y<5) or (z>10))”), removing the parentheses may affect the semantic of the code script. However, since this research focuses on simple conditions (consisting of one or two conditions), applying this rule does not affect the semantic of code segments.

**Print Message Rule:** In this rule, the message part of the print line is removed. Even if print lines provide the same message in terms of meaning, their syntax can be different which may increase the group number in the grouping process. In this research, the asked questions are highlighted print messages. Thus, students provide mostly the same messages based on the
questions requirements. However, even if the meaning of the messages is important, they are removed based on the requirements of the codifying process. Nevertheless, the marker is still able to see the print messages in the marking process. Table 4.6 shows an example of the rule.

**Table 4.6 Example of the print message rule**

<table>
<thead>
<tr>
<th>Before applying the rule</th>
<th>After applying the rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>if (x &gt; 10): print x bigger than 10</td>
<td>if (x &gt; 10): print</td>
</tr>
</tbody>
</table>

**Whitespace Rule:** In this rule, whitespaces are removed before and after numbers, math operators and logic operators. This rule is applicable to any equation in the code segment. If one equation in a code segment consists of one whitespace and the same equation in a different code segment consists of multiple whitespaces, they cannot match. However, they can match if whitespaces are removed by applying this rule. Table 4.7 displays an example of the whitespace rule.

**Table 4.7 Example of the whitespace rule**

<table>
<thead>
<tr>
<th>Before applying the rule</th>
<th>After applying the rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>x = 3 * y</td>
<td>x=3*y</td>
</tr>
</tbody>
</table>

In Table 4.7, whitespaces are removed from the equation after applying the whitespace rule. In this sense, each equation can have a single structure and the similarity ratio between components can be high.

**Swap Rule:** This rule is applicable to the condition part of a code segment if it has more than one condition and consists of only one type of logic operator. The condition part of code segment structures is generally written according to certain rules, such as the first condition being written first, and then logical operators and other conditions being written later. However, the order of the conditions can be different which prevents them from matching even if they are semantically the same. To address this, the order of conditions can be sorted in alphabetical order. Thus, the conditions can be given in a standard form by adopting the swap rule. Non-identical conditions can become identical using this rule. However, this rule does not affect the output of any code script even if conditions can be given in a standard form. Furthermore, this rule is applicable for simple conditions rather than complex ones. Therefore,
this rule can be used in order to increase the group number before the grouping process in accordance with the objectives of this research. The following example demonstrates the application of this rule.

<table>
<thead>
<tr>
<th>Before applying the rule</th>
<th>After applying the rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(y &lt; 4) and (x &lt; 3)</td>
<td>(x &lt; 3) and (y &lt; 4)</td>
</tr>
</tbody>
</table>

In Table 4.8, the order of the conditions is wrong according to the swap rule. The order of the conditions is swapped by applying the rule. After applying the swap rule, the conditions are ordered alphabetically and can be therefore given in a standard form, as shown in Table 4.8.

Despite the fact that most of the variable names were semantically similar as a result of the well-defined questions, they were manually fixed in order to increase the similarity between code segments. This is thus considered as a limitation of this research, which can be resolved using a code writing editor like Scratch (Resnick et al., 2009). Scratch is a drag and drop editor. That is, the coder can drag and drop variable names, control structures, print messages etc. Thus, identical variable names can be captured and this limitation can be addressed.

4.6.3. Grouping Process

Code segments are normalised by applying the generic rules to increase the similarity between them. At the end of the codifying process, each code segment is referred to as a component. In this process, the similarity between components needs to be measured initially, after which the components are grouped based on the similarity measurement ratios. This is because markers provide feedback for only one member from each group in the marking process and the feedback is then copied to mark the rest of the group members. This results in assisting the marker in providing consistent feedback and reducing his/her workload. The grouping process consists of two parts; the similarity measurement and the grouping criteria. Both parts are described in detail in this section.

4.6.3.1. Similarity Measurement

Semi-automated assessment systems described in Section 2.3.1.2 face the same issue of providing inconsistent feedback. Human markers provide feedback based on static assessment
after the dynamic assessment for each code script. This leads to the provision of inconsistent feedback by such systems. This research supports the re-use of markers’ comments for similar code segments (as described in Section 4.5); a process which requires carrying out a similarity measurement. The section is based on the identification of similar components within students’ code scripts. The similarity measurement ratios of components are used in the proposed assessment approach to create groups. Only the grouping process of the semi-automated assessment approach needs to be considered during the development of the similarity measurement environment. However, the codifying process (Process 4) also affects the similarity measurement ratios because it applies the generic rules to increase the similarity between components.

The aim of this section is to explain how the approach measures similarity between components. After the similarity is measured, the similarity ratios are used to create groups. Furthermore, weightings for component parts (described in Section 4.6.1) need to be specified in the similarity measurement formula. In addition, a threshold value needs to be specified when creating groups according to the similarity measurement results. This is due to the fact that without thresholding, rather similar components cannot match perfectly and would be put into separate groups. This results in increasing the marker’s workload due to the increase in the number of groups. Thus, a threshold value is required to place similar code segments in a single group. An experiment was carried out to decide weightings for component parts which will be discussed in this section.

The similarity measurement formula should consist of components’ parts. That is, similarity is a combined calculation which involves calculations for each control structure, condition structure and code segment structure’s similarities. The similarity between two components can be calculated using one of two formulas, namely a sequence type formula and a control type formula. The creation of the similarity formulas can be considered as one of the main contributions of this work, since components are key factors in order to provide consistent feedback, and these formulas are created based on them. If any two components are sequence type components, their similarity cannot be measured using the control type formula. According to the control type formula, two sequence type components’ similarity ratios can be a maximum of 40%. However, two sequence type components’ similarity ratios can of course be more than 40%, but it is impossible achieve a higher percentage using the control type formula. Hence, the sequence type formula was created specifically to measure the similarity between sequence type components. It should be noted that initially, the codifying process
should be applied and these formulas can then be used. In this sense, after the normalisation of data (code segments), the similarity between any components can be measured using the formulas.

\[
\text{Sequence Type Formula} = (Sim_{\text{Code-Segment Structure}})
\]

\[
\text{Control Type Formula} = (Sim_{\text{Control Structure}}) \times 40\% + (Sim_{\text{Condition Structure}}) \times 20\% + (Sim_{\text{Code-Segment Structure}}) \times 40\%
\]

Table 4.9 presents the experimental results. The experiment was carried out to decide weightings for component parts and the value of the threshold. This was done to enable obtaining closely similar components, which directly increases the feedback consistency if they are put into the same group. In light of this, different weightings were used for each component part and the number of similar components was calculated according to different threshold values. The similarity of 55 components was measured and the number of similar components is shown in the table. The numbers displayed in Table 4.9 depend on the sample data used in this research.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Control Structure Weighings</th>
<th>Condition Structure Weighings</th>
<th>Code Segment Structure Weighings</th>
<th>Threshold</th>
<th>Number of Similar Components amongst 55 Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>60%</td>
<td>20%</td>
<td>20%</td>
<td>70%</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>60%</td>
<td>20%</td>
<td>20%</td>
<td>80%</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>60%</td>
<td>20%</td>
<td>20%</td>
<td>90%</td>
<td>39</td>
</tr>
<tr>
<td>Experiment 2</td>
<td>50%</td>
<td>25%</td>
<td>25%</td>
<td>70%</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>25%</td>
<td>25%</td>
<td>80%</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>25%</td>
<td>25%</td>
<td>90%</td>
<td>33</td>
</tr>
<tr>
<td>Experiment 3</td>
<td>50%</td>
<td>30%</td>
<td>20%</td>
<td>70%</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>30%</td>
<td>20%</td>
<td>80%</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>50%</td>
<td>30%</td>
<td>20%</td>
<td>90%</td>
<td>35</td>
</tr>
<tr>
<td>Experiment 4</td>
<td>40%</td>
<td>20%</td>
<td>40%</td>
<td>70%</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>40%</td>
<td>20%</td>
<td>40%</td>
<td>80%</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>40%</td>
<td>20%</td>
<td>40%</td>
<td>90%</td>
<td>27</td>
</tr>
</tbody>
</table>
Well described and short form questions are asked in this research, since students are required to provide their own solutions based on the requirements of the questions. A well described question means that it highlights the code segment type, print messages and condition structures etc. that should be used by students to provide their solutions. In Experiment 4, the weighting of the control structure is less than that in other experiments, and weighting of the code segment structure is greater than that in other experiments. The reason for this is that students generally use the same control structures because of the well described questions and as such, this does not have too much of an effect on the similarity measurement. However, students sometimes provide different code segment structures despite the well described questions specifying how their answers should be structured. Thus, in the research, it was decided to increase the weighting associated with the code segment structure in Experiment 4 compared to the other experiments (see Table 4.9). In addition, the threshold value is accepted as 90% in experiment 4 since closely similar components can be obtained despite the fact that the number of similar component is the smallest among the four experiments. If the threshold value was less than 90% in this research, the number of similar components in the same group would increase, as highlighted in Table 4.9. However, dissimilar code segments would be in the same group, and as such, students would get inconsistent feedback. In this sense, the use of a 90% threshold is crucial for this research in terms of providing consistent feedback for semantically similar components. At the end of the experiment, components that were considered similar were found to be almost identical in terms of structure. In addition, the proposed value of threshold (90%) and weighting of control (40%), condition (20%) and code segment (40%) structures contributed towards providing consistent feedback. These values were chosen based on the experiments’ results. Additionally, the proposed values can be also used for loop (e.g. for, while) statements.

A different set of weightings for component parts could have been used in this experiment. For instance, 33% could have been used for each component part. In this sense, the numbers of similar code segments would increase compared to the used weightings for component parts in this research. However, this would result in lower similarity levels compared to the current similarity measurement results. This is because students already provided rather similar control and condition structures as a result of the well described questions. However, the same could not be said about code segment structures. Thus, the code segment structure’s weighting was increased in order to capture similar code segments. Overall, each experimental parameter was
carefully optimised for this research. Table 4.10 shows three components, of which the similarity among will be calculated step by step.

**Table 4.10 Example of three different components**

| Component 1 | if r <= 0 and h <= 0  
| | print  
| | elif r > 100 and h > 100  
| | print  
| Component 2 | if r <= 0 and h <= 0  
| | print  
| | elif r > 100 and h > 100  
| | print  
| | else  
| | print  
| Component 3 | import math  
| | print  
| | print  

Component 1 and 2 are “else-if” type components while components 3 is “sequence type” component. The rest of this section starts with discussing the measurements of the control structure, condition structure and code segment structure similarities of the components. Then, these measurements are used to calculate the similarity among the components in Table 4.10 using the Sequence or Control Type Formula.

Baxter et al. (1998) measures similarity between code segments using the abstract syntax tree technique. This research adapts Baxter’s formula specifically to measure the control, condition and code segment structure similarities. Shared (identical) nodes are used in Baxter’s formula which refer to the number of identical lines between components (represented by “S”). However, the lines between components should match line by line and their order must be the same. For example, the first three lines between two components can be identical and the rest of the components can be different even if the rest of the components consist of identical lines. In this case, the first three lines can be considered as “S”. On the other hand, the number of different nodes between abstract syntax trees are also used in Baxter’s formula, which refers to number of different lines in component-1 (represented by “L”) and number of different lines in component-2 (represented by “R”).

\[
\text{Component Structure Similarity} = \frac{2 \times S}{2 \times S + L + R}
\]
Where:

\[ S = \text{Number of shared (identical) line between components.} \]

\[ L = \text{Number of different lines in component-1.} \]

\[ R = \text{Number of different lines in component-2.} \]

The whole structure should be matched to measure the similarity. However, even if four or five lines are identical between components consisting of six lines, the similarity could be less than the value of the threshold. In this case, the components cannot be put into same group even though they are closely similar, since this could affect the feedback consistency. Thus, in this research it was decided to use the component structure similarity formula.

- **Similarity Measurement for Control Structures**

This subsection discusses the measurement of the control structure similarity between components. This is the most important part of a component, since this research targets the provision of segmented feedback. Furthermore, the control structure is captured based on the control structure tree illustrated in Section 4.6.1. Table 4.11 displays the control structure parts of the components from Table 4.10.

**Table 4.21 Control structure parts of components**

<table>
<thead>
<tr>
<th>Control Structure 1</th>
<th>Control Structure 2</th>
<th>Control Structure 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>if</td>
<td>if</td>
<td>-</td>
</tr>
<tr>
<td>elif</td>
<td>elif</td>
<td></td>
</tr>
<tr>
<td>else</td>
<td>else</td>
<td></td>
</tr>
</tbody>
</table>

According to Table 4.11, the control structures 1 and 2 have control statements while control structure 3 does not consist of any control statement since it refers to a sequence type code segment. For example, the similarity between control structure 1 and control structure 2 can be measured via the following expression:

\[
\text{Control Structure Similarity} = \frac{2 \times S}{2 \times S + L + R} = \frac{2 \times 2}{2 \times 2 + 0 + 1} = \frac{4}{5} = 80\%
\]

\[ S = \text{Number of shared (identical) lines between components: 2 ("if" and "elif")} \]

\[ L = \text{Number of different lines in component-1: 0} \]
The control structure similarity between components can be measured according to the method presented above. If a component refers to an “if” or ‘loop’ type and another component refers to a “sequential type”, their control structure similarity is not applicable.

- **Similarity Measurement for Condition Structures**

This subsection describes the similarity measurement between the condition parts of components. The condition structure is captured based on the condition structure tree illustrated in Section 4.6.1. Table 4.12 displays the condition parts of the components from Table 4.10.

### Table 4.12 Three different condition structures

<table>
<thead>
<tr>
<th>Condition Structure 1</th>
<th>Condition Structure 2</th>
<th>Condition Structure 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r \leq 0 ) and ( h \leq 0 )</td>
<td>( r \leq 0 ) and ( h \leq 0 )</td>
<td>-</td>
</tr>
<tr>
<td>( r &gt; 100 ) and ( h &gt; 100 )</td>
<td>( r &gt; 100 ) and ( h &gt; 100 )</td>
<td></td>
</tr>
</tbody>
</table>

It is normal for variable names to be different in conditions. However, in these cases the conditions cannot be considered identical, although they may be identical in terms of meaning. Thus, the variable names are fixed in this measurement. This avoids making the condition structure similarity less than the threshold due to the different variable names. In such cases, the general similarity between components is always less than the threshold value. That is, closely similar code segments are put into separate groups which increases the marker’s workload. The similarity between condition structure 1 and condition structure 2 can be measured with the following expression:

\[
\text{Condition Structure Similarity} = \frac{2 \times S}{(2 \times S + L + R)} = \frac{2 \times 2}{(2 \times 2 + 0 + 0)} = \frac{4}{4} = 100\%
\]

\( S = \) Number of shared (identical) line between components: 2 (\( r \leq 0 \) and \( h \leq 0 \) and \( r > 100 \) and \( h > 100 \))

\( L = \) Number of different lines in component-1: 0

\( R = \) Number of different lines in component-2: 0

As can be seen from the measurement result, the condition structure similarity between the two conditions is 100%, which means they are identical. Furthermore, the condition structure similarity between condition structure 1 and condition structure 3 is not applicable due to the fact that component 3 in Table 4.10 is of the “sequential type”.

R = Number of different lines in component-2: 1 (“else”)
• **Similarity Measurement for Code Segment Structures**

Measuring the code segment structure similarity of components involves capturing the code segment structure based on the code segment structure tree illustrated in Section 4.6.1. Table 4.13 displays the code segment parts of the components from Table 4.10.

<table>
<thead>
<tr>
<th>Code Segment Structure 1</th>
<th>Code Segment Structure 2</th>
<th>Code Segment Structure 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>print</td>
<td>print</td>
<td>import math</td>
</tr>
<tr>
<td>print</td>
<td>print</td>
<td>print</td>
</tr>
<tr>
<td>print</td>
<td>print</td>
<td>print</td>
</tr>
</tbody>
</table>

“print” messages are often different due to the different syntax used even if they are identical in terms of meaning. Thus, only print statements (i.e. print commands without messages) are used for this measurement. The similarity between code segment structure 1 and code segment structure 2 can be measured using the following expression:

\[
\text{Code Segment Structure Similarity} = \frac{2 \cdot S}{(2 \cdot S + L + R)} = \frac{2 \cdot 2}{2 \cdot 2 + 0 + 1} = \frac{4}{5} = 80\%
\]

S = Number of shared (identical) lines between components: 2 (“print” and “print”)

L = Number of different lines in component-1: 0

R = Number of different lines in component-2: 1 (“print”)

As can be seen from the measurement result, the code segment structure similarity is 80%. Furthermore, the code segment structure similarity between code segment structure 1 and code segment structure 3 can be measured using the same equation. However, the similarity between the code segment structure 1 and 3 or 2 and 3 is 0%. This is because the first lines are not identical and therefore, do not match. Even if the subsequent lines after the first lines are identical, their similarity is considered to be less than the threshold, as described in this section above.

After the calculation of the three structures’ similarity ratios, they are added together according to the similarity formulas. For example, Table 4.14 shows the similarity calculation (using the control type formula) between component 1 and component 2, which were given in Table 4.10.

70
According to Table 4.14, component 1 and component 2 have a similarity that is less than the threshold (84%). Therefore, they cannot be put into the same group.

A component may refer to an if (if, else-if or else) or a loop (for, while or do-while) type code segment, as illustrated in Figure 4.2. In this sense, the use of loops in the similarity measurement process is similar to the introduced example in this section. In other words, the use of control, conditional and code segment structures of any loop type component (for, while and do-while) is similar to the sequential and if type code segments. As such, sequential, if and loop type code segments similarity can be measured using the similarity measurement formulas described in this section.

As highlighted in Section 4.6.3, the string matching technique is used in the similarity measurement between components. However, a different similarity measurement technique, namely the software metric technique, was also tried before deciding to use the string matching technique in this research. The number of lines in control, condition and code segment structures were measured based on the software metric technique. Then, the numbers were used in order to measure the similarity. However, it was realised that the components considered similar (and are thus put into the same group) did not display high similarity levels. This is due to the fact that the number of lines were used in the similarity measurement instead of matching the lines. In this case, similar code segments would be marked inconsistently after reusing a human marker’s comments for similar code segments. Thus, the string matching technique was used in this research to make use of its support of line-based comparisons to other lines in components. This allows highly similar components to be put into the same group and consistent feedback to be provided.

Finally, it is worth mentioning that it was decided to use the values in Experiment 4 (presented in Table 4.10), including the 90% threshold value, in this research since they provide better results for finding highly similar components. This results in enhancing the consistency of the feedback and benefiting both markers and students.
4.6.3.2. Grouping Criteria

This section discusses the grouping of components according to the similarity measurement ratios. In this process, components are put into the same group if their similarity ratio is more than or equal to the threshold (which is the grouping criterion), otherwise, they are separated in different groups. Code scripts generally include more than one component. For example, if a code script consists of “if” and “loop” type components together, each of them can create a separate group, because these components refer to different code segments. Thus, in this example, two different groups are created by the grouping process. Table 4.15 and 4.16 illustrates two groups: Group 1 and Group 2.

<table>
<thead>
<tr>
<th>Component No</th>
<th>Group 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>for r in range(2,11):</td>
</tr>
<tr>
<td></td>
<td>( sa = (\pi \cdot r \cdot (r + \sqrt{r^2 + r^2}) ) )</td>
</tr>
<tr>
<td></td>
<td>( v = (\pi \cdot r^2 \cdot r)/3 )</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td>2</td>
<td>for r in range(2,11):</td>
</tr>
<tr>
<td></td>
<td>( sa = (\pi \cdot r \cdot (r + \sqrt{r^2 + r^2}) ) )</td>
</tr>
<tr>
<td></td>
<td>( v = (\pi \cdot r^2 \cdot r)/3 )</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td>3</td>
<td>for r in range(2,11):</td>
</tr>
<tr>
<td></td>
<td>( sa = (\pi \cdot r \cdot (r + \sqrt{r^2 + r^2}) ) )</td>
</tr>
<tr>
<td></td>
<td>( v = (\pi \cdot r^2 \cdot r)/3 )</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Component No</th>
<th>Group 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if r&lt;0 or h&lt;0:</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td></td>
<td>elif r&gt;100 or h&gt;100:</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td></td>
<td>else:</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td>2</td>
<td>if r&lt;0 or h&lt;0:</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td></td>
<td>elif r&gt;100 or h&gt;100:</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td></td>
<td>else:</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td>3</td>
<td>if r&lt;0 or h&lt;0:</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td></td>
<td>elif r&gt;100 or h&gt;100:</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
<tr>
<td></td>
<td>else:</td>
</tr>
<tr>
<td></td>
<td>print</td>
</tr>
</tbody>
</table>
Group 1 and Group 2 have five and three components respectively. The first three components’ relative similarity equals or exceeds the threshold, creating Group 1. Likewise, the other three components’ similarity relative to each other is equal to or more than the threshold and as such, they are put into Group 2.

The aim of the grouping process is to create groups of similar (repetitive) components. Repetitive components are used in the marking process to reduce marking time and increase consistency. The correctness of the grouping process depends on the criteria used to measure the similarity between components. In this research, the evaluation of the correctness of the grouping criteria was done visually/manually. This ensures that the components that are in the same group have mostly the same component structures and conditions etc. The grouping process is crucial to the marking process; that is, the workload of the marker directly depends on the number of groups. An effective and accurate grouping process allows the marking process to eliminate the repetitive marking task and enable semi-automation.

4.6.4. Marking Process

Components are grouped in the grouping process, but the code segments are presented to the marker in the marking process to enable him/her to provide reliable feedback. In this process, code segments in a programming code are marked manually or automatically. The marker provides comments for a code segment from each group in manual marking. In automated marking, the marker’s comment is copied to mark the overall code segments in a group. That is, the human marker does not individually provide the comments. Each programming code is therefore marked by the proposed assessment system.

4.6.4.1. Manual Marking

The purpose of the marking process is to provide rich and detailed feedback through the marker. The marker considers the structure of code segments while providing feedback. Table 4.17 shows three code scripts examples including code segments. The code scripts are represented with letters and different colours to facilitate the explanation of the manual marking process.
Each programming code has three code segments. Here, it is assumed that the grouping process has been applied and the components have been grouped. In Table 4.17, code segments are in the same group if their colour is the same. The marker marks only one code segment from each of the three groups in the manual assessment process. For instance, initially, the marker marks Code Script 1 in Table 4.17. In this case, code segments A, D and G are manually marked by the marker. If any code segments in different programming codes are in the same groups as the marked components in Code Script 1, they are marked automatically by the proposed assessment system.

4.6.4.2. Automated Marking

The automatic assessment process marks the rest of the code segments from groups that are not marked. It uses the marker’s comments to mark similar code segments. For instance, it marks code segments B, C, E, F, H and I. Therefore, all code segments (code scripts) are marked through either the manual or the automatic marking process. The re-usability of the marked code segments in the proposed approach is similar to the Case-Based Reasoning (CBR) cycle described in Section 2.3.4.

The CBR cycle can be adopted in this research because using previously marked code segments makes the approach very similar to CBR. Thus, automation can be accelerated in the marking process. For Process 5 (grouping process), components’ structural similarity needs to at least equal the threshold for the components to be put into the same group. However, if a new code segment has a similarity value of less than the threshold with all the cases in the case-base, a marker needs to mark the new code segment. A case means the combination of the code segment and comment (feedback). In contrast, if the similarity value is not less than the threshold, the null adaptation technique is applied. The null adaptation process retrieves the comment from the current case and then applies it to a new code segment. That is, the new code segment is automatically marked according to the CBR cycle. The marking process model
is illustrated in Figure 4.5, which is a more detailed view of the core process (blue rectangle) in Figure 4.1. The marking process model is created based on segmented marking. The similarity measurement is used in the grouping process in this marking process model, which refers to Process A in Figure 4.5. Process A also refers to the RETRIEVE process in the CBR cycle described in Section 2.3.4.
Fig. 4.5 Marking process model
Process B in Figure 4.5 represents the grouping process in Figure 4.1. On the other hand, processes C and D together refer to the marking process in Figure 4.1. To elaborate, Process C refers to manual marking and Process D refers to automatic marking. As shown in Figure 4.5, the manual and automatic marking processes appear as parallel processes. However, they are not parallel processes, as can be seen from Figure 4.1. The process in Figure 4.5 initially focuses on marking automatically which depends on the similarity measurement ratios. If the similarity between a code segment and its nearest neighbour is greater than or equal to the threshold, the new code segment is automatically marked in Process D (which refers to the REUSE process in the CBR cycle) by using the null adaptation technique. In other words, the new code segment is put into the retrieved case’s group. The nearest neighbour means that if the similarity between a code segment in any case and the new code segment is highest, the case is considered as the nearest neighbour. Alternatively, Process C takes place where the human marker manually marks the code segment.

In addition, the marked code segments can be revised by human markers if required in Process E (which refers to the REVISE process in the CBR cycle). In this case, markers may add additional comments or remove comments to provide comprehensible and helpful feedback for novice students. The marked code segment is then retained in Process F which refers to the RETAIN process in the CBR cycle. The retained code segment may be re-used to mark new code segments based on similarity measurement ratios. In such a case, new code segments can be automatically marked through Process D and as such, consistent feedback can be provided using the proposed marking process model. All code segments in Process G can be used as cases in the case-base cycle described in Section 2.3.4.

4.7. Conclusion

This chapter presented a new semi-automated assessment framework for programming code scripts. Novice programmers’ solutions were observed and subsequently analysed. The results of the analysis showed that students’ solutions consist of similar code segments. A new marking technique was created and called “segmented marking”. A new marking model was also developed based on the segmented marking technique and adapted to CBR cycle in order to provide consistent feedback for similar code segments. The most critical process in the presented framework is the grouping process. This is because placing dissimilar code segments into the same group can result in students obtaining inconstant feedback, which may hinder the
improvement of their programming skills. This motivated the use of the string matching technique in the similarity measurement part of the grouping process as it offers accurate identification of similar code segments, which paves the way towards the provision of consistent feedback.

Segmented marking is a novel way of marking. Thus, a feasibility study of the segmented marking technique was carried out and is presented in Chapter 5. Furthermore, details of the improvement of the marking process model based on the findings of the feasibility study are presented in Chapter 6.
CHAPTER 5
Feasibility Study of the Marking Technique

5.1. Introduction

The proposed semi-automated approach described in Chapter 4 advocates commenting on each code segment in a code script, a process referred to as segmented marking. An initial model has been developed based on the proposed segmented marking technique in Chapter 4. This chapter examines the feasibility of the proposed marking technique. The outcomes of this chapter will be used as user interface design requirements for the marking tool. Furthermore, this chapter provides the details of the pilot experiment and the main experiment carried out to check the feasibility of the proposed marking technique. The aim of both experiments was to measure the feasibility of the proposed marking technique. The pilot experiment was carried out initially, and revealed some shortcomings on the marking technique before moving to the main experiment. The shortcomings were then addressed before the main experiment was conducted. Moreover, data collected from the experiments was analysed to ascertain whether the participants were satisfied with providing feedback based on the use of the segmented marking technique. Due to the fact that this research aims to provide consistent feedback based on segmented marking, students’ thoughts on the generated feedback were not captured.

This chapter has three sections. The first two sections provide details about the pilot experiment and main experiment such as their aims, participants details, marking page layout designs, data collections, results and findings. The third section focuses on the user interface design requirements of the marking tool.

5.2. Pilot Experiment

This experiment is conducted to examine the feasibility of the proposed marking technique and the applicability of the marking page layout (illustrated in Figure 5.1.) of the marking tool, which is outlined in Section 5.2.5. Participants can provide comments based on the segmented marking approach thorough the proposed marking page layout. After the experiment, participants were questioned on their opinions on the proposed marking technique.
5.2.1. Pilot Experiment Aim

The aim of the experiment is to check the feasibility of the proposed marking technique in light of the following objectives:

1. Find out whether the marking page layout (Figure 5.1. in Section 5.2.5.) enables participants to provide segmented marking.
2. Establish whether participants provide detailed feedback through segmented marking.
3. Find out whether the participants are satisfied with the proposed marking technique based on the proposed assessment approach.

Each objective stated above is important for this research. However, the third objective holds a special significance. The reason is that if participants are satisfied with providing feedback based on segmented marking, the proposed semi-automated assessment approach may work.

5.2.2. Participants

Four participants were used in the pilot experiment. The participants were PhD students with Python programming experience and who knew how to make comments since they already had experience in marking programming modules. Two of them were also teaching assistants in a Python programming module. However, they were not-expert participants. However, this research applies the purposive sampling for the main experiments since finding an expert participant who has at least ten years’ experience on marking of structural programming languages is difficult. The participant’s experience is quite important in measuring the feasibility of segmented marking. According to Ofqual (2014), expert participants must have an undergraduate degree in the subject area. However, the existing literature lacks clarity regarding an expert participant’s experience (in terms of how many years of experience are required). As such, participants with marking experience of at least ten years are accepted as expert participants in this research in order to achieve reliable results. Furthermore, at least five participants are required for the main experiment (Connelly 2008; Creswell 1998). Connelly (2008) states that the sample size in a pilot study/experiment should be 10% of the main experiment. In light of this requirement, four participants were deemed sufficient for the pilot experiment.
5.2.3. Experiment Questions

Three different questions were asked of 55 students undertaking a first semester (2014) “introduction to programming” module at a UK University, who provided, in response, their code scripts. Code scripts in this research refer to the programming code and consist of code segments, as highlighted in Section 4.3. The questions are presented in Table 5.1. A maximum of two code segment types were sufficient when providing code scripts for each question, such as ‘segmentation’ (question 1), ‘segmentation and elif’ (question 2) and ‘segmentation and loop’ (question 3) code segments types. These three types of code segments were described in Section 4.6. The 55 students provided code scripts for the questions, resulting in a 165 code scripts in total. However, since participants rely on paper-based feedback in this experiment which could make the marking process tedious and time consuming, it was decided to reduce the number of code scripts for this experiment. Students generally provide similar code segments for each question in terms of code segment types. Thus, code segments that were sufficiently different were chosen to reduce number of code segments. In this manner, the number of code scripts were reduced from 165 to 17. While having a larger number of code scripts is certainly possible, an increase in the number of code scripts could lead to participants having to make comments for quite similar code segments which could be time-consuming and unnecessary repetition. Furthermore, the following statements were important factors in choosing code scripts for this experiment:

- **Code Segments:** Code segments were chosen according to the code segment structure. Many novice programmers have difficulty in providing the correct code segment in terms of the control structure, which was described in Section 4.6.3. On the other hand, the majority of students tend to provide correct code segments in terms of structure when faced with well described questions, as was highlighted in Section 4.6.3.

- **Programming Errors:** Different code segments were chosen for this study because they exhibit different error types such as syntax and logic errors, which were explained in Section 2.2.3. For instance, many equations in the code segments do not consist of parenthesis which may cause logic programming errors. In addition, students’ solutions often tend to have problems with indentation which may cause syntax errors.

- **Comments and Code Layout:** Code segments chosen for this study contained sufficient comments, a few comments or no comments. In other words, some of code segments were well commented which helped to understand the code easily, while
others either did not consist of any comment, or consisted of a few comments. Code segments were also chosen according to the code layout in terms of either well-formatted and poorly-formatted code segments. Well-formatted code layout means that the code segment can be easily followed and understood. On the other hand, poorly-formatted code layout cannot be easily followed and is often difficult to understand.

Table 5.1 *Questions used to collect novice students’ solutions.*

<table>
<thead>
<tr>
<th>Experiment Questions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Question 1:</strong></td>
<td></td>
</tr>
<tr>
<td>The surface area of a cone is given by</td>
<td></td>
</tr>
<tr>
<td><strong>Surface Area</strong> = ( \pi r(r + \sqrt{h^2 + r^2}) )</td>
<td></td>
</tr>
<tr>
<td>The volume of a cone is given by</td>
<td></td>
</tr>
<tr>
<td><strong>volume</strong> = ( \frac{\pi r^2 h}{3} )</td>
<td></td>
</tr>
<tr>
<td>Write a program which asks the user to enter integer values for the radius of the base and the height of the cone. The program should then calculate the surface area and the volume of the cone. The program should print a user-friendly message telling the user the value of the surface area and the volume <strong>on one line</strong>. The units for ( r ) and ( h ) are in centimetres. The following is an example message: The surface area is: 76.64 cm(^2) and the volume is: 41.89 cm(^3).</td>
<td></td>
</tr>
<tr>
<td><strong>Question 2:</strong></td>
<td></td>
</tr>
<tr>
<td>Make a copy of q1.py and save it as q2.py. Modify q2.py so that after the user has inputted both the radius and height values, the program checks for the following conditions before calculating the area:</td>
<td></td>
</tr>
<tr>
<td>If the radius or the height is not a positive number then the program should print the message:</td>
<td>ERROR: Both the ( r ) and ( h ) values must be positive numbers!</td>
</tr>
<tr>
<td>If the radius or the height more than 100 then the program should print the message:</td>
<td>ERROR: Both the ( r ) and ( h ) values must be less than or equal to 100 cm!</td>
</tr>
</tbody>
</table>

*You MUST utilise the “if..elif..else” statement in your code.*
Question 3:
Create a new program to display a table with 3 columns:

a. In column 1, list all of the possible integers from 2 to 10.
b. In column 2, display the surface area of a cone whose height and radius are both equal to the values in column 1 (i.e. \( h = r = \text{column 1} \)).
c. In column 3, display the volume of a cone whose height and radius are both equal to the values in column 1 (i.e. \( h = r = \text{column 1} \)).

For example, the first row should calculate the surface area and the volume of a cone with height=2 and radius=2.

The calculated surface area and volume must be to two decimal places and line up with the decimal point.

Your output should resemble the following:

<table>
<thead>
<tr>
<th>Height/Radius</th>
<th>Surface Area</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30.34</td>
<td>8.38</td>
</tr>
<tr>
<td>10</td>
<td>758.45</td>
<td>1047.20</td>
</tr>
</tbody>
</table>

You MUST use a loop to do this. The volume and the surface area columns must use a calculation to find the answer.

5.2.4. Marking Page Layout Design

A marking page layout was designed in the pilot experiment based on the proposed assessment approach and is illustrated in Figure 5.1. A code segment marking area was created for each code segment on the right side of the marking page layout. The reason behind this is that it enables participants to provide comments near the relative code segment, thus facilitating the segmented marking process that this research supports. As such, feedback can be made more detailed and supportive for novice programmers. Furthermore, a general marking area was also created at the end of the marking page layout for general comments. According to proposed semi-automated assessment approach, the participant’s comments are used to comment on similar code segments to the marked code segment. However, the participant may also need to make general comments. In this case, if the participant uses the segmented marking areas to make general comments, incorrect feedback may be provided since the general comment is also used for similar code segments. Thus, a general marking area is also created. Finally, empty spaces were also created between code lines in the code segments on the left side of the marking page layout (labelled as “annotation marking area” in Figure 5.1) to enable participants to provide annotations. Comments made in the general and the code segment marking areas and annotations made in the annotation marking area are altogether considered feedback in this research. Although the proposed assessment approach mainly advocates segmented marking, participants are free to use any marking area to provide feedback in the
marking page layout. Furthermore, comments made in the general marking area are not considered as segmented marking. If participants mostly prefer segmented marking, the feasibility of the proposed assessment approach improves. Moreover, annotations can also be made in the annotation marking area to make the feedback more helpful for novice programmers. In this sense, annotations are also considered beneficial in providing helpful and detailed feedback (Hwang et al., 2008). Hence, the three marking areas in marking page layout were designed to understand which marking area is mainly preferred by the participants.

In the marking page layout, the marking areas’ size vary based on the code segments’ size. For example, the size of the annotation marking area depends on the code segments’ height and length. The length of the annotation marking area size is also bigger than the length of the code segment marking area, in order to display code segments to participants legibly. In such a case where a long code line in a code segment is used, it can be displayed in two lines. The general marking area also enables participants to provide sufficient comments of four or five lines length.

<table>
<thead>
<tr>
<th>Annotation Marking Area</th>
<th>Code Segment Marking Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Segment - 1</td>
<td>Comment…</td>
</tr>
<tr>
<td></td>
<td>__________________________</td>
</tr>
<tr>
<td>Code Segment - 2</td>
<td>Comment…</td>
</tr>
<tr>
<td></td>
<td>__________________________</td>
</tr>
</tbody>
</table>

**Fig. 5.1 Example of the marking page layout in the pilot experiment**

Additionally, the labels of the marking areas are illustrated to easily explain the marking areas in Figure 5.1. Participants were informed that they could provide feedback using the three marking areas as they saw fit. For example, they may provide a general comment and/or a code segment comment and/or annotation. However, they were not informed about the importance
of segmented marking, or the areas designed to support this approach in order to avoid biased results.

5.2.5. Data Collection

Data was collected in two ways in the pilot experiment. Initially, the participants commented on the students’ code scripts through the proposed marking page layout illustrated in Figure 5.1. Secondly, an interview was conducted with each participant after the marking process. This research advocates segmented marking and the re-usage of the participants’ comments for the repetitive code segments; a process which was described in Section 4.5. However, participants were not made aware of the importance of segmented marking in this research. If participants do not prefer the code segment marking area in the marking process, the approach put forward would not work as well. Thus, the interview aims to get the participants’ opinions about the segmented marking based on the approach even if they do not prefer segmented marking.

5.2.6. Results and Discussion

This section presents the three main results of the pilot experiment, which are the preferred marking areas of the participants, the categorisation of the participants’ comments and their responses to the interview. This section also presents a discussion about interpreting the participants’ opinions.

5.2.6.1. Preferred Marking Areas

The four participants provided comments for 17 students’ code scripts. The code scripts consisted of 29 code segments in total. In this case, each participant could provide 17 general comments for the 17 code scripts. In addition, each participant could provide annotations in the annotation marking area and comments in the code segment marking area for the 29 code segments separately. Table 5.2 shows which marking areas were chosen by the participants in the marking process.
Table 5.2 Numbers of preferred marking areas

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Code Segment</th>
<th>General</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant 1</td>
<td>12/29</td>
<td>-/29</td>
</tr>
<tr>
<td>Participant 2</td>
<td>-/29</td>
<td>19/29</td>
</tr>
<tr>
<td>Participant 3</td>
<td>1/29</td>
<td>29/29</td>
</tr>
<tr>
<td>Participant 4</td>
<td>8/29</td>
<td>13/29</td>
</tr>
<tr>
<td>Total Number of Used Marking Area</td>
<td>21/116</td>
<td>61/116</td>
</tr>
</tbody>
</table>

As shown in Table 5.2, all four participants preferred the use of the general marking area. On the other hand, the annotation and the code segment marking areas were preferred by three participants. This result was not an expected result for this research and the reason behind it will be discussed in Section 5.2.6.3.

Generally, comments made in the general marking area are suitable for providing basic feedback, but are not important for this research. The proposed assessment approach refers to formative assessment, which was described in Section 2.2.1. Segmented marking can be more effective in providing helpful and rich feedback for novice programmers based on the formative assessment. Moreover, comments made in the general marking area cannot be reused as easily as comments made in the code segment marking areas. The proposed assessment approach supports the re-use of comments for repetitive code segments, as described in Chapter 4. However, the number of repetitive code scripts is mostly less than the number of repetitive code segments. The reason is that code scripts are mostly bigger than code segments in terms of size. Although comments made in the general or the code segment marking areas can be reused based on the proposed assessment approach, repetition of the comments made in general marking area may reduce feedback consistency and increase the participants’ workload. Hence, repetition of the comments made in the code segment marking area is regarded as a better approach than repetition of the comments made in the general marking area. In this case, the proposed assessment approach provides more detailed feedback through segmented marking. More detailed information will be provided in Section 5.2.6.3 which will discuss the participants’ opinions about the use of the marking areas in the proposed marking page layout.

The annotation marking area is the least preferred marking area, as shown in Figure 5.2. The aim of the annotation is to add a short remark on the code (Hwang et al., 2016). Table 5.2
provides information on the number of annotations that were provided by the four participants. Moreover, the four participants made annotations using different annotation styles such as:

- Text (T): provide annotation text only.
- Underline (U): underline the code and then provide annotation text.
- Tick and Cross marks (TC): put tick or cross sign near the code and then provide annotation text.
- Arrow reference (A): draw an arrow and then provide text.

The participants provided these styles on their own and were not restricted into making any choices. The numbers of provided annotation texts according to annotation styles are illustrated in Table 5.3.

### Table 5.3 Numbers of annotations used by the participants.

<table>
<thead>
<tr>
<th>Annotation Type</th>
<th>T</th>
<th>U</th>
<th>TC</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant 1</td>
<td>7</td>
<td>11</td>
<td>-</td>
<td>2</td>
</tr>
<tr>
<td>Participant 2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Participant 3</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>Participant 4</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>6</td>
</tr>
<tr>
<td>Total Number of Annotation</td>
<td>11</td>
<td>11</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 5.3 illustrates that the four participants provided 33 annotations in total using the four annotation types. T (Text) and U (Underline) were the most commonly used annotation types while TC (Tick and Cross marks) was the least commonly used annotation type. Even if annotations can be reused for other students’ code scripts, there should be identical content between students’ code scripts to safely repeat the annotation. Code segments mostly do not consist of identical lines of code. They may have different syntax even if they are similar in terms of control structure. Hence, the participants’ workload may increase due to the high number of non-identical code lines in the code segments. Furthermore, the proposed assessment approach may not provide consistent feedback due to the different syntax used in the code lines. One of the challenges of the paper-based and the semi-automated assessment systems is to provide consistent feedback. If participants provide annotations for code lines which are not identical, this may result in inconsistent feedback. Thus, annotations are optional in this research. In this case, participants can be make annotations only if they deem the annotation necessary.
5.2.6.2. Categorisation of participants’ comments

Participants’ comments made in this experiment were categorised based on a criterion which is the ‘meaning of the comments or annotations’. The reason is that the participants’ comments can be reused based on the proposed assessment approach if they use the same or similar comments for repetitive code segments. Although the comments are categorised based on their meaning, they can generally be divided into generically reusable, specific or superficial comments.

- **Meaning of the comments:** Participant comments could be different but in essence provide the same message in terms of semantic meaning, such as ‘good use of loop’ and ‘loop is fine – well done!’.
- **Generically reusable comments:** a participant makes comments which can be used for more than one code segment, such as ‘The structure is good’.
- **Specific comments:** a participant provides detailed comments such as ‘The loop is correct but the range is wrong’.
- **Superficial comments:** a participant provides weak comments which are not generally helpful for students such as ‘seems good’.

The participants made numerous generically reusable comments which are illustrated in Appendix B. Furthermore, although their comments were phrased differently, they mostly have the same meaning. In this sense, these comments can be reused for repetitive code segments based on the proposed assessment approach which reduces their workload and enables providing consistent feedback.

5.2.6.3. Participants’ opinions from interview

Questions were asked to the participants in this experiment based on the proposed assessment approach. The proposed marking page layout refers to a new marking technique which the participants were not aware of. Thus, the pilot experiment mainly aimed to obtain the participants’ opinions about the feasibility of the marking technique. Each participants’ response was noted and each is presented in the section.

*Question:* Which marking area was the most helpful for you in the marking page layout to provide detailed and helpful comments?
Rationale: This question was asked to find out whether the code segment marking area is the most helpful marking area for the participants. Participants’ opinions are shown in Table 5.4 below.

Table 5.4 Participants’ opinion about the most helpful marking area.

<table>
<thead>
<tr>
<th>Participant No.</th>
<th>Participant’s Opinion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“The code segment marking area was the most helpful marking area to provide detailed comments but I mostly preferred the general marking area to complete marking process in a short time.”</td>
</tr>
<tr>
<td>2</td>
<td>“The code segment marking area.”</td>
</tr>
<tr>
<td>3</td>
<td>“The code segment area was the most helpful. I made comments for each code segment using the code segment marking area. The code segment marking area facilitated a way to provide detailed comments.”</td>
</tr>
<tr>
<td>4</td>
<td>“The code segment marking area. However, I mostly preferred to make comments using the general marking area. The reason is that manual marking is tedious and time-consuming.”</td>
</tr>
</tbody>
</table>

Looking at Table 5.4, it is apparent that although the participants’ thought that the code segment marking area was the most helpful marking area to provide detail and helpful feedback, they mostly preferred the general marking area. According to their opinions, manual marking is tedious and time-consuming, which is why they mostly chose to make comments in general marking area.

Question: Which marking area was the least helpful for you in the marking page layout in terms of providing detailed comments?

Rationale: This question was asked to find out whether the general marking areas are required in the marking page layout to provide detailed and helpful feedback for novice programmers. Participants opinions are shown in Table 5.5.
Table 5.5 Participants’ opinion about the least helpful marking area.

<table>
<thead>
<tr>
<th>Participant No.</th>
<th>Participant’s Opinion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“Even if I mostly preferred the general marking area, it was the least helpful marking area for providing detailed remarks. However, it was the easiest way for me to provide feedback.”</td>
</tr>
<tr>
<td>2</td>
<td>“General marking area.”</td>
</tr>
<tr>
<td>3</td>
<td>“The general marking area was the least helpful for me. However, it saved my time.”</td>
</tr>
<tr>
<td>4</td>
<td>“The general marking area was the least helpful in terms of providing detailed comments. “</td>
</tr>
</tbody>
</table>

One can see from Table 5.5 above that although the participants’ felt that the general marking area was the least helpful marking area in terms of providing rich and detailed feedback, they mostly preferred to use it. Furthermore, three of the participants highlighted that comments made only in the general marking area cannot be sufficient for novice programmers.

**Question:** If you should prefer only two marking areas from the three marking areas in the marking page layout, which would you choose to provide detailed and helpful comments?

**Rationale:** This question was asked to find out the participants’ thoughts about the three marking areas. They may consider that one or two marking area could be sufficient to provide feedback instead of the three marking areas. Alternatively, they may consider that the proposed three marking areas are necessary for providing detailed and helpful feedback. The participants’ opinions are shown in Table 5.6.
It can be inferred from the table above that participants believed that the code segment and annotation marking areas are sufficient for providing detail and helpful feedback.

**Question:** Do you think the marking areas’ size in marking page layout is important for providing enough comments?

**Rationale:** This question was asked to find out participants’ thoughts about the size of the marking areas. The participants’ comments can be too long and overflow from the marking area into the marking page layout. In this case, the participants may think that the marking area should be unrestricted in order to make detailed comments freely. Participants’ opinions are shown in Table 5.7.

<table>
<thead>
<tr>
<th>Participant No.</th>
<th>Participant’s Opinion</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“I could prefer the code segment and the annotation marking areas.”</td>
</tr>
<tr>
<td>2</td>
<td>“The code segment and the annotation marking areas.”</td>
</tr>
<tr>
<td>3</td>
<td>“The code segment and the annotation marking areas could be sufficient for providing detailed feedback.”</td>
</tr>
<tr>
<td>4</td>
<td>“The code segment and the annotation marking areas.”</td>
</tr>
</tbody>
</table>

From the table above, one can see that participants thought the marking areas should be unrestricted in order to provide comments freely.
Question: What is your suggestion(s) if you think the marking areas in the marking page layout are not suitable to give detailed and helpful feedback for novice programmers?

Rationale: This question was asked to find out the participants’ suggestions on providing more detailed and high-quality feedback in the proposed marking page layout. The participants’ opinions are listed in Table 5.8 below.

<table>
<thead>
<tr>
<th>Participant No.</th>
<th>Participant’s Suggestions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“The marking page layout is fine but the general marking area can be removed from the marking page layout. The code segment and the annotation marking area can be enough for providing detailed comments.”</td>
</tr>
<tr>
<td>2</td>
<td>“Comments made in the general marking area can be helpful for programmers having experience on computer programming. However, the annotation and the code segment marking areas can be better than the general marking area for providing detailed comments for novice programmers. In this case, the participant may deeply focus on each code segment to provide detailed comments through the annotation and the code segment marking areas.”</td>
</tr>
<tr>
<td>3</td>
<td>“The general marking area was the least helpful marking area. Thus, it can be removed from the marking page layout. In this manner, the participants can provide more detailed comments.”</td>
</tr>
<tr>
<td>4</td>
<td>“The marking page layout is fine but the general marking area can be preferred for non-novice programmers. Novice programmer needs to receive more detailed comments which can be provided using both the annotation and the code segment marking areas.”</td>
</tr>
</tbody>
</table>

Table 5.8 indicates that the participants generally thought that the code segment and the annotation marking areas are sufficient for providing feedback for novice programmers.
5.2.6.4. Interpreting Participants Opinions

The participants’ comments were analysed according to the comment types which were described in Section 5.2.6.2. Table 5.9 presents information on the provided comment types based on the marking areas.

<table>
<thead>
<tr>
<th>Annotation Marking Area</th>
<th>Code Segment Marking Area</th>
<th>General Marking Area</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generically reusable comment</td>
<td>0</td>
<td>67%</td>
<td>33%</td>
</tr>
<tr>
<td>Specific Comments</td>
<td>53%</td>
<td>42%</td>
<td>5%</td>
</tr>
<tr>
<td>Superficial Comments</td>
<td>3%</td>
<td>25%</td>
<td>72%</td>
</tr>
</tbody>
</table>

This research aims to provide generically reusable and specific comments through segmented marking. However, the four participants mostly preferred the general marking area. According to the participants’ responses, the general marking area is the easiest way to provide feedback. On the other hand, they indicated that the code segment marking area is the most suitable marking area for providing rich and helpful comments. In light of this, the pilot experiment could be carried out with a marking page layout that does not contain a general marking area. In such a case, the percentage of the generically reusable and specific comments (see Table 5.9) in the code segment marking area could be higher. Furthermore, the generically reusable comments made in the general marking area could be made in the code segment marking area instead of general the marking area. Thus, the participants’ comments could be repeated based on the proposed assessment approach which supports the re-use of comments for the repetitive code segments.

The participants also highlighted that the marking areas’ size is quite important. Their comments sometimes overflowed the marking areas. The marking areas’ size for each code segment in the marking page layout could be fixed based on the longest code segments’ size. For example, the length of each code segment area (box) can be made to have the same size in the marking page layout. In this manner, the code segment marking area could be used more effectively by the participants. Moreover, the participants could provide more generically
reusable and specific comments through the extended code segment marking area. In addition, the participants could easily make comments close to the relevant code through the extended marking areas. This should make the feedback more easily understandable for novice programmers and help them fix their errors on the control structure, programming errors etc.

5.2.6.5. Findings

Findings from the pilot experiment are listed below.

1. The participants mostly made comments using the general marking area. However, the preference of the general marking area was not an expected result for this research since this research supports segmented marking. The participants stated that manual marking (paper-based assessment) is tedious and time-consuming, and as such, they mostly preferred the general marking area to provide comments in a speedier fashion.

2. According to the interview results, the participants were generally satisfied with the use of the annotation and the code segment marking areas in the marking page layout for providing generically reusable and specific comments for novice programmers. This shows that the proposed assessment approach can be applied to provide specific and generically reusable comments through the code segment and annotation marking area in the marking page layout. In this sense, the comments made in the code segment marking area can be reused for repetitive code segments based on the proposed assessment, which was an expected result for the pilot experiment.

3. The pilot experiment revealed that the marking page layout needed to be redesigned and an experiment should be carried out with the redesigned layout. The number of comments made in the code segment marking area could thus increase using the redesigned marking page layout and the proposed assessment approach can function better. Furthermore, feedback can be more consistent and the participants’ workload can be reduced.

5.3. Main Experiment

The main experiment was carried out with a re-designed marking page layout. The marking page layout was re-designed based on the findings from the pilot study. The main reason of the re-designing of the marking page layout is to increase the preferred code segment marking area in this experiment. This experiment shares the same aims and objectives with the pilot experiment, which were described in Section 5.2.1.
5.3.1. Participants

Twelve participants (who were different from the participants in the pilot experiment) provided feedback in the experiment. They consisted of PhD students with Python programming experience and knew how to provide comments since they had experience on marking programming modules. Three of them were teaching assistants in a Python programming module and two of them were teaching assistants in a C++ programming module. The use of twelve participants in this experiment was justified in Section 5.2.2.

5.3.2. Questions

The majority of the questions that were used for the pilot experiment were also used for the main experiment, since both experiments share the same aims and objectives. However, two code scripts (consisting of three code segments in total) used in the pilot experiment were not used for this experiment since they were closely similar with another code script used in the pilot experiment. In addition, some general comments could be also used for dissimilar code segments. Thus, closely similar code segments were not used to highlight the reusability of some generic comments for dissimilar code segments. Furthermore, an additional code script was added for this experiment. The added code script was different from the other code scripts in terms of code structure. In this sense, with the added code segments, the participant can provide more comments in total in this experiment. Thus, the reusability of comments for similar code segments and the importance of the segmented marking for this research can be more strongly highlighted based on the proposed assessment approach. The described statements to choose code scripts (in Section 5.2.3.) for the pilot experiment were also used for the added code script in this experiment.

5.3.3. Marking Page Layout Design

The marking page layout illustrated in Section 5.2.4. was re-designed based on the findings of the pilot experiment. The re-designed marking page layout is illustrated in Figure 5.2. It consists of two marking areas, namely the annotation and the code segment marking areas.
The general marking area was removed from the marking page layout. Furthermore, each marking area’s size in a code script was fixed to the largest code segment’s size in the marking page layout, so that the participants can provide comments in the code segment marking area instead of the now removed general marking area. The participants of the pilot experiment mostly made generically reusable comments using the general marking area. The participants of this experiment could make generically reusable comments by utilising the code segment marking area which has sufficient space to make comments in contrast to the limited space that was available in the pilot experiment. Furthermore, the participants’ generically reusable comments can be also re-used for different students’ code scripts based on the proposed assessment approach. These changes should effectively help the proposed assessment approach to function well.

The proposed assessment approach described in Chapter 4 supports marking of repetitive code segments using of the participants’ comments made in the code segment marking area. This serves to provide consistent feedback and reduce participants’ workload through the use of the case-based reasoning cycle, which was described in Section 2.3.4. In this sense, if participants mostly prefer segmented marking, the experiment can better demonstrate the feasibility of the proposed marking technique. Furthermore, annotations could still be important even if participants mainly prefer segmented marking. The reason is that the addition of annotations to segmented marking makes the feedback more detailed and helpful for the novice programmers based on the formative assessment, as was highlighted in the pilot experiment.
5.3.4. Data Collection

In this experiment, data was collected via two methods. Initially, the participants commented on the students’ code scripts using the re-designed marking page layout. Secondly, a questionnaire was distributed to each participant after they commented on the code scripts in order to capture their thoughts on the segmented marking process and the marking page layout. Since the aim here was to measure the feasibility of the segmented marking process, which is a novel way of marking, only the participants’ thoughts were captured to learn whether they approved of the segmented marking technique.

5.3.5. Results and Discussion

This section presents two main results, namely which marking area in the marking page layout was preferred by the participants and the participants’ responses to the questionnaire. This section also presents an interpretation of participants’ responses and findings.

5.3.5.1. Preferred Marking Areas

Participants made comments for each code scripts including 30 code segments in total. Two code scripts (consisting of 3 code segments) were removed and a code scripts (consisting of 4 code segments) was added for this experiment, as described in Section 5.3.2. Thus, 30 code segments were used in this experiment whereas 29 were used in the pilot experiment. Table 5.10 shows which marking areas were chosen by the participants in the marking process.

<table>
<thead>
<tr>
<th>Participant No.</th>
<th>Annotation marking area</th>
<th>Code Segment marking area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant 1</td>
<td>-/30</td>
<td>29/30</td>
</tr>
<tr>
<td>Participant 2</td>
<td>22/30</td>
<td>24/30</td>
</tr>
<tr>
<td>Participant 3</td>
<td>28/30</td>
<td>28/30</td>
</tr>
<tr>
<td>Participant 4</td>
<td>30/30</td>
<td>2/30</td>
</tr>
<tr>
<td>Participant 5</td>
<td>9/30</td>
<td>20/30</td>
</tr>
<tr>
<td>Participant 6</td>
<td>27/30</td>
<td>27/30</td>
</tr>
<tr>
<td>Participant 7</td>
<td>-/30</td>
<td>28/30</td>
</tr>
<tr>
<td>Participant 8</td>
<td>27/30</td>
<td>20/30</td>
</tr>
</tbody>
</table>
The table above highlights that the code segment marking area had a higher preference among the participants than the annotation marking area, where all participants (12) preferred the code segment marking area, while three of the participants did not prefer the annotation marking area. This is preferable since the proposed assessment approach supports the re-use of comments made in the code segment marking areas to comment on repetitive code segments based on the formative assessment. Around 75% of the comments provided in the code segment marking areas were generically reusable while approximately 13% of the comments were specific comments and 12% were superficial comments. On the other hand, 63% of the annotations made in annotation marking areas were specific comments while 31% were generically reusable comments and 6% were superficial comments. While feedback quality is important for students (especially novice programmers), segmented marking is the focus of this research. The feedback quality also completely depends on the participant’s experience (Ala-Mutka, 2005). The participants’ responses to the questionnaire will be discussed in Section 5.3.6.2 and interpreted in Section 5.3.6.3.

The participants also made annotations using different annotation styles, which were described in Section 5.2.6.1. The numbers of the provided annotations made in the annotation marking area are presented in Table 5.11.

<table>
<thead>
<tr>
<th>Annotation Type</th>
<th>Participant 1</th>
<th>Participant 2</th>
<th>Participant 3</th>
<th>Participant 4</th>
<th>Participant 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>-</td>
<td>13</td>
<td>29</td>
<td>22</td>
<td>8</td>
</tr>
<tr>
<td>U</td>
<td>-</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>TC</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>-</td>
<td>7</td>
<td>17</td>
<td>27</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.11 Numbers of annotations used according to type.
Twelve participants made a total of 282 annotations in the annotation marking areas. The **T** (Text) annotation type was the most used annotation type. Further detailed explanation of use of annotation will be provided in Section 5.3.5.3 following the participants’ responses to the questionnaire.

### 5.3.5.2. Participants’ response to the Questionnaire

Questions were asked to examine the effect of the segmented marking on providing feedback for novice programmers. Furthermore, the questions were also designed to investigate which marking area provides the best opportunity for making the feedback more detailed. Finally, the participants’ thoughts on the re-usability of comments to provide feedback for similar code segments were captured. Participants’ responses will contribute towards the identification of the user interface design requirements of the marking tool and measuring the feasibility of the marking technique.

**Question: Which of the following statements most closely applies to you?**

- **a)** Providing feedback via the code segment marking areas is more effective for me than using the annotation marking areas on the code itself.
- **b)** Providing feedback via the annotation marking areas is more effective for me than using the code segment marking areas on the code itself.
- **c)** Despite using the code segment marking areas, the annotation marking areas are still important for providing detailed and helpful feedback.
- **d)** Despite using the annotation marking areas, the code segment marking areas are still important for providing detailed and helpful feedback.
Rationale: This question was asked to find out whether the participants agree that the code segment marking area is more important than the annotation marking area, despite the fact that they are used together to provide feedback.

Fig. 5.3 Participants response on the marking areas

As shown in the figure above, the majority of the participants (75%) consider that both marking areas should be utilised in the marking process. Generally, the participants’ response for this question can be considered as a positive result for this research, as segmented marking was deemed to be more important than annotations according to the majority of the participants. Furthermore, although 25% of the participants consider that annotations are more important than comments made in the code segment marking areas, they are not against the use of the code segment marking area.

Question: Which of the following statements most closely applies to you?

a) I believe that annotations made in the annotation marking area should only take a single form (for example underlining) to ensure clarity and understandability.

b) I believe that annotations made in the annotation marking area should take many forms (underlining, tick boxes, arrows etc.) to convey complex information and ensure clarity and understandability.

c) None of the above.

Rationale: This question was asked to find out whether a single form of annotation is efficient for providing easily understandable feedback.
As illustrated in Figure 5.4, half of the participants considered that one type of annotation is sufficient to make feedback understandable. On the other hand, the other half of the participants thought that more than one type of annotations should be utilised to make the feedback easily comprehensible. The proposed assessment approach suggests the re-use of comments made in the code segment marking area. Furthermore, the re-use of annotations is also possible based on the proposed assessment approach but may increase the workload of the participants. The reason is that code lines need to be identical to ensure the correctness of the annotations. In this case, one type of annotations can be simpler and may help provide easily comprehensible feedback for novice programmers.

**Statement:** I believe that in order for a comment to be unambiguous, the distance between the comment and the code itself should be minimised.

**Rationale:** This statement was asked to find out if the distance between the comment and code segment should minimised in the marking page layout. The participants’ responses are illustrated in a bar chart in Figure 5.5.
Looking at Figure 5.5, it is apparent that the majority of the participants (66%) at least agree with the above statement while 17% of the participants disagree. The participants who disagree with the statement may prefer to provide comments anywhere in the marking sheet according to their own marking approach. However, Clark (2011) states that feedback can be more helpful if the distance between the code and comments is limited. Furthermore, since the majority of the participants already at least agree with the statement, the distance should therefore be minimized, especially considering that this approach is suggested for providing feedback for novice programmers.

**Statement: I believe that the code segment marking area needs to be unrestricted in length, as the comment length is independent of the code segment length.**

**Rationale:** This statement was asked to find out whether the participants agree about making the marking areas unrestricted in length to help make comments more freely. The participants’ responses are illustrated in a bar chart in Figure 5.6 below.

![Fig. 5.6 Participants response on the length of the comment area](image)

The figure above demonstrates that around 60% of the participants at least agree with the statement. On the other hand, the rest of the participants hold a neutral view on the statement. Overall, no participant disagrees with the statement. In light of this, the length of the comment area should be unlimited to allow participants to provide comments in an easy manner.

**Statement: I find that many of the comments I make are generic, or common enough that an efficient mechanism to write them once and use them many times without rewriting them would be valuable.**

**Rationale:** This statement was asked to find out whether the participants agree about the automatic reuse comments for repetitive code segments based on the proposed
assessment approach. The participants’ responses are shown in a bar chart in Figure 5.7.

![Bar chart showing participants' responses on the use of comments for similar code segments]

**Fig. 5.7 Participants’ responses on the use of comments for similar code segments**

As shown in Figure 5.7 the majority of the participants (around 91%) at least agree with the statement while the rest of the participants hold a neutral view. The participants’ responses on this statement is an encouraging result since the proposed assessment approach supports the reuse of comments for similar code segments based on the CBR cycle. In this manner, participants can provide more consistent feedback and reduce the marking workload.

### 5.3.5.3. Interpreting Participants’ Responses and Findings

Findings of the main experiment are listed and interpreted below.

1. The experiment showed that the participants are satisfied with the proposed marking page layout. It also indicated that the code segment marking area is the most effective marking area for providing helpful and detailed feedback for novice programmers. In the pilot experiment, the code segment marking area was not the most preferred marking area. However, its emergence as the most preferred area in the main experiment is an expected result. This is due to the removal of the general marking area from the marking page layout which made participants prefer the code segment marking area for making comments.

2. Most of the participants (91%) agreed on the re-use of comments for repetitive code segments. The finding shows that the proposed marking technique is feasible. In this sense, participants can provide consistent and timely feedback to novice programmers through segmented marking.

3. According to the experiment, the annotation marking area is still important even if the code segment marking area is the most effective marking area. By utilising the
annotation marking area, participants can provide richer and more detailed feedback. However, using the annotation marking area may also result in decreasing feedback consistency and increase the marking time. Code lines can be repeated based on the proposed assessment approach. However, code lines must be identical to ensure the correctness of the feedback, which is not typically the case. As such, it can be suggested that participants make annotations only if they consider them useful for enhancing the feedback quality, which could save the marking time spent.

4. The results of experiment indicate that the distance between a comment and a code should be minimised to make feedback easily understandable. Furthermore, if more than one annotation type is used to provide helpful and detailed feedback, the distance between the code and the comment cannot be minimised. In light of this, the use of one annotation type (such as to add text notes) may make the feedback simpler and easy understandable, as well as allow the distance to be minimised between the code and the comment.

5. The experiment showed that marking areas should not be restricted. Some of the participants’ comments made in both the annotation and the code segment marking areas overflowed the marking areas in the experiment. As such, the size of the marking areas should be unrestricted to prevent this issue.

The proposed marking technique is feasible since the participants’ responses to the questionnaire are expected results based on the proposed assessment approach. The experiment also contributes towards identifying the user interface design requirements for the marking tool, which will be described in Section 5.4. The participants provided comments for code segments in both the pilot and the main experiments. Some of the code segments equated to nested code segments. The meaning of a nested code segment was described in Section 4.6.1 and the proposed assessment approach supports marking of nested code segments. However, in this study, participants made comments for whole code segments rather than for the nested part of the code segment and the remaining part separately. This may stem from the fact that the participants might have thought that the code segments are already short and easily understandable and did not need to provide separate comments. Furthermore, the proposed assessment approach supports providing feedback for novice programmers, who often provide short code scripts since they are in the early learning phase of their programming experience. Even if the participants provided comments using a semi-automated assessment tool, they might still find the process to be tedious since code segments are short and they must provide
comments for the nested part of code segments. In light of this, marking of nested code segments cannot be considered as a user interface design requirement in this research, although the approach generally supports marking of nested code segments.

The study showed that the participants made more comments in the code segment marking area of the new marking page layout than in the code segment marking area of the initial marking page layout (illustrated in Section 5.2.4), the reason of which was described in Section 5.3.5.1. One of the objectives of this research is to provide detailed feedback for novice programmers. Bearing this in mind, if a participant sees only a code segment instead of a code script, they may provide more detailed comments. The marking idea is different from the traditional full-marking method on which the development of the initial marking model was based on in Chapter 4. This new marking approach is referred to as partial marking in this research. The initial model will be also developed based on partial-marking in addition to full-marking in order to determine which marking approach is more effective. In this sense, partial-marking can be also considered as a user interface design requirement and the feasibility of the partial marking can be evaluated through the marking tool.

5.4. User Interface Design Requirements of the Marking Tool

The user interface design requirements are listed below:

- Two separate user interfaces in order to provide feedback through the full and partial marking techniques.
- The user interface must support the provision of feedback based on segmented marking (e.g. be able to comment and annotate on code segments).
- The user interface must support the reuse of comments for repetitive code segments based on the case-based reasoning (CBR) cycle.
- The user interface must minimise the distance between comments and code segments to make the feedback easy understandable.

The requirements described above were derived based on the results of both the pilot and the main experiments, with a particular focus on the results of the more refined main experiment.

5.5. Conclusion

This chapter described the conduction of a pilot experiment and a main experiment as means to evaluate the feasibility of the proposed segmented marking technique and identify the user
interface design requirements of the marking tool. A marking page layout was designed for main experiment, and according to the results, the most preferred marking area was the code segment marking area, which was an expected outcome of the main experiment. In addition, most of the participants (91%) approved of the re-use of their comments for repetitive code segments within the same group. In this sense, consistent feedback can be provided based on the segmented marking technique. Thus, segmented marking can be considered feasible based on this research’s purposes.

According to the observations made in experiments, the participants were more focused on code segments when code script consisted of only a single code segment. In this sense, a completely different marking technique was revealed based on segmented marking, which is partial marking. The next chapter explores the design and development of the marking tool. The chapter provides details of the improvements made on the proposed initial marking model (discussed in Chapter 4) based on the use of the full and partial marking techniques as marking models.
CHAPTER 6
Design and Development of the Marking Tools

6.1. Introduction

The proposed semi-automated approach is based on the segmented marking technique which potentially enables the provision of consistent and personalised feedback. Segmented marking means marking of each code segment in a code script, as described in Chapter 4. The feasibility of the marking technique was evaluated in Chapter 5. Furthermore, a set of design requirements of the user-interface for the marking tool were also identified in section 5.4 based on the results of the feasibility study. According to results of the feasibility study, two user interfaces need to be developed to implement the requirements. This chapter introduces the design and development of the user interfaces. The requirements consist of two types of marking environments, namely full-marking and partial-marking. In addition, the marking process model (introduced in Section 4.6.4) needs to be developed according to a set of requirements. This chapter therefore particularly focuses on the development of the marking models for the marking environments and user interfaces. The full and partial marking models are analysed in two different sections. They support the use of code segments and some of the CBR terminology (described in Section 2.3.4) is used in the development of the models.

This chapter consists of five sections. Section 6.2 discusses the order of use of the user interfaces. Section 6.3 provides an overview of the marking system, while Section 6.4 focuses on the design requirements of the user interfaces. Section 6.5 describes the marking processes models. It also discusses the case definition, case generation and case usage. The final section (Section 6.6) presents the development process of the prototype user-interfaces for the full and partial marking tool.
6.2. Order of Use for User Interfaces

Two different user interfaces are developed based on the full and partial marking approaches. Chapter 5 demonstrated that both approaches have advantages and disadvantages. The full-marking approach is the more traditional way of marking, and offers an advantage for markers in that they can link between code segments in the marking process. In this sense, the provided feedback can be detailed and useful for improving novice programmers’ programming skills. On the other hand, the advantage of the partial marking technique is that markers can dedicate their attention on code segments in the marking process since they only see a code segment instead of the code script in partial-marking. This allows markers to provide more detailed comments for the novice programmers. The disadvantage of the partial-marking is that markers cannot link between code segments in the marking process which may affect the feedback quality. Partial marking is a new technique and as such, markers can gradually change their habits to fit the new marking style. The two user interfaces should support these approaches and the marker can choose one of them to provide feedback. However, this study focuses on evaluating the effectiveness of the two marking approaches. This is because since this study compares the two new marking environments based on segmented marking, completion of the whole marking process with either the full or partial marking user interface can be more beneficial for this research. For instance, if the marker choses the full-marking technique, they need to complete the marking process with the full-marking technique. In this sense, they can start with the full marking approach since it supports the traditional way of marking, where the full code scripts can be seen. The marker can subsequently use the partial marking tool and start to benefit from it. In addition, order of use for user-interfaces is up to the users. Alternatively, in future studies, the marker can start the marking process using the partial marking user interface and continue using the full marking approach.

The two interfaces are developed to first check whether markers/participants accept and approve their use. Furthermore, both user interfaces are assessed in order to establish which is more beneficial by collecting markers’ feedback on them. A full evaluation of the user interfaces is conducted in Chapter 7.
6.3. Overview of the Marking System

The initial marking process model was illustrated and described in Section 4.6. This model reuses markers’ comments for similar code segments by adopting the CBR cycle to provide consistent feedback. Section 2.3.4 described the CBR cycle and Section 4.6.4 explained how a comment is reused based on the CBR cycle.

The new marking technique utilises segmented marking which enables the marker to provide comments for each code segment in a code script. It intends to reduce the repetition in the marking process. In this sense, a marking tool supporting the re-use of markers’ comments reduces the repetition in the marking process and enables the marker to provide consistent feedback. The marker’s comments and the code segment are stored in the case-base as a case. In the context of this study, a case means that the code segment and the comment constitute a case. The case can then be retrieved from the case-base to comment on repetitive code segments. In this manner, the marker’s comment can be adopted to the repetitive code segments through the marking tool.

Figure 6.1 presents an overview of the marking system. The marker provides comments and the marking system subsequently re-uses the comments to mark similar code segments. Finally, feedback is generated.

![Fig. 6.1 Overview of the marking system](image-url)
As illustrated in Figure 6.1, the scheme suggests that the human marker and the machine (computer) collaborate to provide consistent and personalised feedback. Part A in this system refers to manual marking, while Part B refers to automated marking. Both parts require a separate user interface. In this sense, Part B needs to be also used based on partial marking in addition to full marking in user interfaces.

6.4. Design Requirements

The user interface design requirements were identified based on the results of the feasibility study described in Section 5.4. There are four main requirements for the user interface:

1. Two separate user interfaces in order to provide feedback through the full and partial marking techniques.
2. The user interface should provide feedback based on segmented marking (i.e. enables the provision of comments and annotations on code segments). Parts A and B (in Figure 6.1.) together both implement segmented marking since a human marker or a computer provides comment(s) for code segments in the both parts.
3. The user interface should to adapt markers’ comment based on the CBR cycle for similar code segments (i.e. enables the re-use of markers’ comments for similar code segments). Additionally, the user interface should suggest the most suitable case’s comment to accelerate automation in the marking process.
4. The user interface should minimise the distance between the code segments and the comments.

The full and partial marking processes have different marking models. These models are developed based on the marking process model (illustrated in Section 4.6.4.) and the user interface design requirements.

6.5. Marking Process

The marking process of the semi-automated approach was briefly described in Chapter 4. The process utilises the segmented marking technique. This section describes the development of the full and partial marking processes. CBR terminology (described in Section 2.3.4) is used in the both the full and partial marking process models in order to link between the process models and the CBR cycle. Although these models support different methods of marking, both of them advocate segmented marking in the marking process.
6.5.1. Full-Marking Process

This subsection describes the full-marking process model, which is developed based on the user interface requirements described in Section 6.4. The full-marking process is similar to traditional marking which displays students’ code scripts on screen. Figure 6.2 displays the full-marking process model. It consists of five areas, namely Area A, B, C, D and E. In full marking, the marker needs to see the code script when providing comments on any code segment in a code script. As such, the marker initially selects a question and then chooses a student’s code script and provides comments for the code segment(s) in the code script (requirement 2 in Section 6.4). The code segment and the comment(s) constitute a case which is stored in the case-base. In this process, the most suitable case (Nearest Neighbour) is retrieved from the case-base, which refers to Area A (the retrieve process in the CBR cycle) in the Figure 6.2. If the similarity ratio between the code segment in the case and the current code segment is less than the threshold value (which is set at 90%, as described in the Section 4.6.2), the marker manually provides comments. On the other hand, if the similarity ratio is greater than the value of the threshold, comments provided for the case are suggested for the code segment. If marker approves the suggested comment, the code segment is commented using the suggested comment. Otherwise, the marker manually provides a different comment. The marking of the code segment refers to Area B - manual marking - in Figure 6.2 and to Part A in Figure 6.1.

Another requirement is to apply the marker’s comments for repetitive code segments (requirement 3 in Section 6.4). This is similar to the adapting process from CBR. After the code segment is commented, repetitive code segments are automatically marked re-using the comment. This process refers to Area C (automated marking) in Figure 6.2. In such a manner, the marker’s comment is adopted for repetitive code segments based on the CBR cycle.

The marked code segments need then to be reviewed by the marker to ensure the correctness of the automatically reused comments. Area D refers to the review process in Figure 6.2. If the marker approves the reused comment, the comment and code segment constitute a case. Otherwise, the most suitable case is retrieved from the case-base to accelerate the review process. If marker approves the suggested comment, it is used to modify the re-used comment. Alternatively, if the marker disapproves the suggested comment, he/she must manually modify the re-used comment in the review process.
Finally, the reviewed code segments are put into the case-base, which refers to Area E (Retain process in the CBR cycle). The final user interface requirement is minimizing the distance between the code segment and comment. In this case, the marker provides comments close to (i.e. in the vicinity of) the code segment, which minimises the relative distance as required. This process is repeated until all code segments (all students’ code scripts) are reviewed.
Fig. 6.2 Full-marking process model
As highlighted in Section 6.1., although partial marking is a new way of marking, it also supports segmented marking, as does full marking. The next subsection discusses the partial marking process in detail.

### 6.5.2. Partial-Marking Process

The partial marking process model is presented in this subsection and, as the full marking process model, is also developed based on the user interface design requirements (described in Section 6.4.). The partial-marking process utilizes a new method of marking in which a code segment is displayed instead of a whole code script. The reason is that markers can dedicate more focus on the code segment in the marking process and the comments they provide can be more detailed than in the traditional way of marking. Furthermore, the correctness of the re-used comments is important. If the marker is satisfied by the re-used comments for the code segments, the new method of marking can be used in many CBA systems in future. Figure 6.3 illustrates the partial-marking process model. The model consists of two parts namely, Part-I and Part-II.

In the new way of marking, the marker needs to only see a code segment. Thus, in Part-I, the marker initially selects a question and the system subsequently displays only a code segment to the marker. Part-I consists of Area A, B and C which are the same as Area A, B and C in the full marking process model. That is, Area A, B and C refer to retrieve process, manual marking (Part A in Figure 6.1) and automatic marking (Part B in Figure 6.1) respectively. This manual marking process is repeated until each displayed code segment is commented. Furthermore, the comments provided for a code segment are adopted (re-used) for repetitive code segments through the Area C process. The marker must complete Part-I to move to Part-II.

In Part-II, the marking tool combines the comments to generate feedback for each student’s code script. The marker then reviews the generated comments since the automatically commented code segment can be inapplicable, which refers to Area D (review process). The marker can begin to review any code segments in a code script. In this process, if marker does not approve of the generated comments for a code segment, they modify the comment. Alternatively, the comment is approved. This means that the marker does not need to take any action when approving any comment in this model. However, the user interface could be implemented to ask the marker in terms of approving each automatically generated comment. In such a case, the marker has to check each generated comment, as otherwise, in the current
version of the user interface, the marker may not read each generated comment. In this sense, the approval of the generated comments can be considered as future work to this research.

Finally, the reviewed code segments are saved by the system as a new case in the case-base, which refers to Area E (retain process). Thus, the saved case can be retrieved and re-used to comment new code segments in future. Furthermore, the saved case can help provide consistent and useful feedback. Part-II is also repeated until each generated comment is reviewed.
Fig. 6.3 Partial-marking process model
As it can be seen from Figure 6.2 and Figure 6.3, a case is generated and can be re-used if the marker approves the comment of the case. The rest of this subsection briefly discusses case definition, generation and usage. A case can be represented with the following notation:

\[ \text{Case} = \text{CASE A (Code Segment A, Comment A)} \]

From the notation above, it can be inferred that a case consists of a code segment and a comment. In a case, a code segment refers to the problem and a comment refers the solution. In addition, each case’s size can be different and each is stored in the case-base.

Figure 6.4 shows the case generation process. Initially, a human marker comments on a code segment and then the code segment and the comment create a case. The human marker’s comment is re-used to comment on repetitive code segments. In this process, the marker reviews the created cases. If the human marker approves any case(s), they are stored in the case-base, which finishes the case generation process.

1. cs is displayed
2. Human marker provides C for the cs
3. Case is created e.g. CASE (cs, C)
4. C is reused automatically for the repetitive code segments of cs
5. Human marker reviews the automatically created CASEs
   IF human marker accepts any automatically created CASE, THEN the CASE is generated and stored in case-base.

- cs is code segment
- C is comment(s)

**Fig. 6.4 The case generation process**

The number of cases depend on the code segments obtained from the code scripts. The generated cases can be re-used to comment on new code segments. A new code segment refers to one that has not been commented manually or automatically so far. As shown in Figure 6.2 and Figure 6.3, cases are retrieved to comment on a new code segment, which enhances the semi-automated marking process. Figure 6.5 illustrates the case usage process. As shown in Figure 6.5, a CASE consists of a code segment and a comment. Initially the most suitable (Nearest Neighbour) case (code segment, comment) is retrieved from the case-base if the similarity between the code segment in the case and the new code segment is greater than the
threshold. The comment in the CASE is then automatically re-used for the new code segment. A human marker reviews the automatically commented code segment in terms of the correctness and applicability. If they approve the comment for the code segment, the retrieved case from the case-base is used. This finalises the usage process of the generated case is used.

<table>
<thead>
<tr>
<th>NN CASE (cs, C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Retrieving NN CASE (cs, C) from case-base</td>
</tr>
<tr>
<td>2. If sim between the cs and new_cs is okay,</td>
</tr>
<tr>
<td>C is re-used to provide comment for the new_cs</td>
</tr>
<tr>
<td>3. Human marker reviews the re-used C</td>
</tr>
<tr>
<td>If the human marker approves the re-used C</td>
</tr>
<tr>
<td>C is adopted to the new_cs and therefore the retrieved case is used</td>
</tr>
<tr>
<td>A NEW_CASE is created eg NEW_CASE (new_cs, C)</td>
</tr>
<tr>
<td>Else</td>
</tr>
<tr>
<td>Human marker manually provides a C or modifies the C.</td>
</tr>
</tbody>
</table>

- NN is Nearest Neighbour
- cs is code segment
- C is comment
- sim is similarity
- new_cs is new code segment

**Fig. 6.5 The case usage process**

The generated cases can be used to provide feedback for many new code segments. The new code segments do not need have the same questions. After applying the generic rules in the codifying process described in Section 4.6.2.; if similarity between the nearest neighbour case and any new code segment is greater than the threshold, the comment of the nearest neighbour case can be used. In this sense, new code segments from different questions can be also commented. This helps enhancing the automation of the marking process as well as providing consistent feedback.
6.6. Development of the prototype user-interfaces

This section describes the development of the full and partial marking tools’ user interfaces based on the full (Figure 6.2) and partial (Figure 6.3) marking process models respectively. The user interfaces are implemented using web and database technologies. Full marking refers to the traditional method of marking. In contrast, the partial marking tool advocates a new way of marking. Both tools support segmented marking. Markers can provide personalised and consistent feedback using the full-marking and partial marking tools. However, the full-marking tool can be more useful than partial marking tool, as the markers see the code scripts when they are providing feedback for code segments. Thus, they can link between code segments and provide more beneficial feedback. In contrast, markers cannot make any links between code segments in the partial marking tool, as they only see code segments in the marking process, as described in Section 6.5.2. Since this research targets the provision of feedback for novice programmers’ solutions for short answers questions, markers may not need to link between code segments. In this context, the partial-marking tool can also be greatly beneficial.

6.6.1. Full-Marking Tool

One of the requirements for a user-interface is to enable segmented marking (requirement 2 in Section 6.4). This refers to the marking of each code segment in a code script rather than providing only general comments for the code script. Furthermore, adapting the marker’s comments to mark repetitive code segments refers to requirement 3 in Section 6.4. Thus, the full-marking tool’s user interface (see Figure 6.6.) is designed to accommodate these requirements. This user-interface has four panes, namely **general information**, **code segment**, **comment** and **suggestion**.

The **general information pane** is placed at the top side of the screen and provides information about the question name and shows the percentage completion indicator. The indicator is not a requirement for the tool but it is developed as way of encouraging the marker in the marking process. The “previous” and “next” arrows are used to easily move to the previous or next student’s code script.

The code script is displayed to the marker in the **code segment pane** in Figure 6.6. The marker can start commenting on any code segment in the code script. Some code lines may be quite
long and be displayed in two lines. Thus, the code segment pane is designed to be wide compared to the comment and the suggestion panes so that each code line is displayed in a single line. Furthermore, code lines are coloured since this helps the marker to recognise the important parts of the code segment such as the code structure, print messages and comments etc. In this sense, the marker can easily focus on assessing the code segment and providing comments on the important parts. In addition, scrolling down is not used in this pane since the marker can focus better on the code segment if the full code segment is displayed on the screen without needing the scroll down function. However, if this user interface is further developed in future to provide feedback for expert markers’ code scripts, the scroll down function should be used since they may provide longer pieces of code. However, since this research is concerned with the assessment of novice programmers’ code scripts, short pieces of code are typical and the scroll down function is not needed.

The comment pane is positioned between the code segment and the suggestion panes. The rationale behind this stems from the fact the marking technique advocates the marking of code segments close (near) to the code. Thus, this positioning helps to minimise the distance between the comment and code which can make the comments more comprehensible by novice students. The comment pane could be placed to the left side of the user interface. However, the distance between the code and comment cannot be minimised in such a case. In this pane, the marker needs to provide comments and subsequently press the ‘insert’ button. The use of buttons in many user interfaces of marking tools is commonplace and facilitates the marking process for markers.

Lastly, the suggestion pane is developed to help the marker in the marking process. The most suitable (Nearest Neighbour) case is retrieved from the case-base and comment of the case is displayed into the suggestion pane, which refers to Area A (retrieve process) in the full-marking process model illustrated in Figure 6.2. In this sense, the suggested comment can be used to comment on the code segment by clicking the suggested comment. Otherwise, the marker must manually type comments to provide feedback. The marking of the code segment refers to Area B (manual marking) in Figure 6.2.
Different colours are used in the comment pane. If a code segment is not marked, it is displayed in red as illustrated in the comment pane. If a code segment is marked, it is indicated by green. The comment of the marked code segment is also re-used to mark the repetitive code segments, which refers to Area C (automatic marking) in Figure 6.2. A marked code segment is displayed in Figure 6.7.

Figure 6.7. illustrates an example in which 20% of the code segments have been marked and 80% of the code segments need to be marked to complete the marking process. After a code segment is marked manually or automatically, the marker can modify the comments using the ‘Save’ or ‘Save All’ buttons. The ‘Save’ button is used to modify the code segment only while the ‘Save All’ button is used for repetitive code segments. The aim behind the creation of the ‘Save All’ button is that if a marker provides an incorrect comment for the repetitive code segments, they can modify the comment to provide appropriate feedback using this button.
As shown in Figure 6.7, a code segment is marked and its comment pane is green. Each code segment must be commented on to complete the marking process. Figure 6.8 displays a marked code script. The percentage value in the percentage completion indicator increases after the marker provides comments for each code segment. When the indicator reaches 100%, the marking process is completed and each code segment is displayed in green as illustrated in Figure 6.8. Furthermore, the marker reviews the automatically marked code segments to ensure the correctness of the re-used comments, which refers to Area D (review process) in Figure 6.2. The comment panes of each automatically marked code segment are also displayed in green.
In the Area D process in Figure 6.2 (review process), the marker does not need to take any action anything if they approve the re-used comment. Otherwise, the marker can modify the comment in two approaches. The first approach applies the retrieve process and the comment of the most suitable case is displayed in the suggestion pane to modify the current comment. The approach way involves the manual modification of the comment by the marker if they do not approve the use of the suggested comment. The reviewed comments and code segments are put into the case-base as a new case, which refers to Area E (retain process) in Figure 6.2.

At the end of the marking process, the tool also presents a report, an example of which is showed in Figure 6.9. The provided comments for repetitive code segments needs to be consistent. Although the creation of the report is not a design requirement, it is created to provide detailed and key information on the marking exercises such as the number of marked code scripts and code segments. The detailed information provided by the report on the two marking processes (full and partial) can then be compared in order to highlight the most effective marking process.

<table>
<thead>
<tr>
<th>REPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exam No</td>
</tr>
<tr>
<td>Question No</td>
</tr>
<tr>
<td>Number of Code Script</td>
</tr>
<tr>
<td>Number of Code Segment</td>
</tr>
<tr>
<td>Number of Manually Marked Code Segment</td>
</tr>
<tr>
<td>Number of Automatically Marked Code Segment</td>
</tr>
<tr>
<td>Number of Edited Code Segment</td>
</tr>
</tbody>
</table>

**Fig. 6.9 An example of a report on the marking process**

The illustrated report in Figure 6.9 also informs the marker on the number of the manually and automatically edited (modified) code segments. The marker can also provide feedback using the partial marking tool, the user interface development of which is described in the following subsection.
6.6.2. Partial-Marking Tool

The partial marking process model illustrated in Figure 6.3 consists of two parts, namely Part-I and Part-II. The partial-marking tool (Part-I) user-interface is displayed in Figure 6.10. In Part-I, only one code segment is displayed from the repetitive code segments since the provided comment for the displayed code segment is re-used to comment on the repetitive code segments. The main differences between the full marking tool and Part-I is that the full marking tool displays the code script while the Part-I displays a code segment, as described in Section 5.3. The marking process of the code segment in this part is the same as in the full-marking tool. Initially, the most suitable case is retrieved and the comment of the case is displayed in the suggestion pane, which refers to Area A (retrieve process) in Figure 6.3. If the marker approves the suggested comment, it is used to comment on the code segment. Otherwise, the code segment is commented on manually, which refer to Area B (manual marking) in Figure 6.3. After the comment is shown in the comment pane, the marker needs to press the ‘submit’ button. The tool then re-uses the marker’s comment to comment on the repetitive code segments, which refers to Area C (automatic marking) in Figure 6.3. That is, the comments are reused to provide feedback on the repetitive code segments.

Sample scripts about ‘if’ type code segments were illustrated in Section 6.6.1. In this section, sample scripts about the ‘for loop’ and ‘while loop’ type code segments are illustrated, while more sample scripts are also presented in Appendix E.

![Partial-Marking Tool User Interface](Fig. 6.10)

The partial-marking tool (Part-I) user-interface

When the percentage completion indicator reaches 100%, the partial-marking tool combines the manually or automatically commented code segments in Part-II to generate comments for each code script. In this case, the marker can check the correctness of the generated feedback.
The partial-marking tool (Part-II) user-interface is illustrated in Figure 6.11. The marker can accept the comment or modify it to provide better feedback, which refers to Area D (review process) in Figure 6.3. If the marker chooses to approve the generated comment, they need to click the ‘Accept’ button. Alternatively, if the marker chooses to modify the generated comment, they should click the ‘Save All’ button after the comment is modified.

Initially, the comment pane is displayed in red which indicates that the comment has not been reviewed. After the marker accepts or modifies the generated comments, the comment pane is displayed in green which indicates that the comment has been reviewed. Furthermore, the comment panes of all repetitive code segments are coloured green after the marker accepts or modifies the generated comments, which accelerates the review process. Figure 6.12 illustrates the accepted or modified comments in this part.
Even if the code segments are displayed in green, the marker can modify the reviewed comment for only one code segment using the ‘Save’ button or can modify the comments for all repetitive code segments using the ‘Save All’ button in the same way as in the full-marking tool. The reviewed comment and code segment are put into the case-base as a new case, which refers to Area E (retain process) in Figure 6.3. Finally, the tool presents a report on the marking process which is also similar to the report illustrated in the Figure 6.9.

This research strongly suggests the reuse of comments provided in the comment pane based on the semi-automated assessment approach. According to the feasibility study results described in Section 5.3.5.2, participants felt that annotations are still important despite segmented marking being the most important aspect of assessment for them. As such, an annotation plugin (Scrible Legacy Toolbar) is used based on the feedback received from the feasibility study. Different annotation techniques could be chosen instead of the plugin. For instance, pens could be used to provide annotations. However, their use is not common and most markers (academic staff) use computers (desktop or laptop) to save time. Another alternative would be to use a computer mouse to make annotations. However, computer mouses are mostly used for image annotation and it is not easy to use them to make annotations, which is why most people prefer not to use them for programming solutions (Russell et al., 2008). Another reason for their unsuitability for this task is that a marker would need to use the mouse to click around the boundary of a line or text in a programming solution, which could often prove to be an inefficient and frustrating task. Thus, the annotation plugin was found to provide the best option for making annotations since it allows marker to make notes, highlight, change text colour and save the annotations. Figure 6.13. displays the annotation toolbar (Fu et al, 2004). The annotation toolbar is used to make annotations for code line(s) in the code segment area.

![Scrible Legacy Toolbar](image)

**Fig. 6.13 Annotation toolbar**

To make an annotation, markers need to click the yellow text icon in the toolbar. The marker can then highlight any code line to make the annotation. After, the marker highlights the code line, the annotation box becomes visible and the marker types the annotation. Figure 6.14 illustrates annotated code lines using the toolbar. Two lines have been annotated, but the
The marker can make unlimited annotation using the toolbar. Furthermore, the marker can choose the colour of the annotation as desired.

**Fig. 6.14 Annotated code lines**

Figure 6.14 shows a marked code script consisting of two code segments. The CBR cycle has been adopted to comment on repetitive code segments in both tools. Markers are allowed to provide comments and annotations through the developed prototype user-interfaces and the annotation toolbar.

The marking tools reuse markers’ comments to mark repetitive (similar) code segments. However, the order of a manually marked code segment and an automatically marked code segment can be different in their own code scripts. This could potentially lead to incorrectly marking the automatically marked code segments if their order is different to that of the manually marked code segment. To avoid this, different colours are used to provide correct feedback. When a code segment is manually marked, its and repetitive code segments’ comment panes are coloured green if their order is same in their own code scripts. Otherwise, the repetitive code segments’ comment pane is displayed in yellow. In this case, the marker needs to be careful in the reviewing process to ensure the provision of appropriate feedback for code segments in the yellow coloured comment pane.

Even if two code segments’ order is same in their own code scripts, the number of code segments in their own code scripts could be different. For example, two similar code segments can be in the second order (i.e. appear as the second code segment) in their own code scripts. However, one code script may consist of four code segments while the other consists of two code segments. In such a case, after a code segment is manually marked, the repetitive code
segments’ comment pane is displayed in yellow. This alerts the marker to the need to double check the correctness of the comment in order to provide appropriate feedback.

When the marker modifies any automatically marked code segment in the reviewing process, other automatically marked code segments are also modified if the manually modified and automatically modified code segments meet some criteria. The criteria are:

- The code segments’ (manually and automatically marked) order must be the same in own code scripts.
- The number of code segments in code scripts must also be the same.

Meeting the criteria above ensures the provision of more consistent feedback and the reduction of the marker’s workload is reduced. Otherwise, the marker would need to modify each marked code segment manually. This could result in the same shortcomings of the paper based assessment approaches described in Section 2.2.2.

In the manual marking process, a marker can provide feedback for a code script consisting of three code segments (such as Code Segment A, B and C). Their comments are re-used to mark similar or repetitive code segments in different code scripts. On the other hand, a different code script can consist of two code segments (such as Code Segment B and C). In this case, code segment B and C are automatically marked based on the proposed assessment approach. However, the marker may need to provide feedback for the missed code segment A. However, this is not possible through the user interfaces of the full and partial marking tools. This is because the marker’s comment on code segment A made in the comment pane of code segment B or C could be repeated for similar code segments with the code segment B or C. Thus, students may receive incorrect feedback as a result. This represents a limitation for this research. To address this limitation, markers can make additional checks to detect those cases and address them. This limitation can therefore be mitigated in future research studies.

6.7. Conclusion

This chapter presented the development of user interfaces of the both the full and partial marking tools. The marking process models of the full and partial marking techniques were described in detail. Both models have been developed based on segmented marking and adopted the CBR cycle. Despite the fact that both marking process model support segmented marking, human markers can link between code segments in the full marking approach since
the whole code script is displayed to them instead of a code segment. In this sense, the full marking approach is closer to the traditional way of marking than the partial marking approach. Thus, human markers may prefer the full marking approach as a method of providing consistent feedback in a short time period, since, unlike the case of the partial marking approach, they do not have to spend time in the review process. The next chapter is therefore dedicated to discussing the evaluation of the implemented marking tools in order to decide which marking approach is more beneficial for human markers.
CHAPTER 7
Marking Tools Evaluation

7.1. Introduction

The full and partial marking tools were developed and implemented based on the user interface design requirements described in Section 6.4. Furthermore, the Case-Based Reasoning (CBR) cycle was adapted within the proposed marking approach described in Chapter 4. This chapter presents an experiment which examines the feasibility of the marking tools in terms of providing feedback based on formative assessment. It also provides information on the time-saving and consistency of the feedback. This chapter consists of five sections. Section 7.2 highlights the aim of the experiment, whereas Section 7.3 discusses the experiment’s design. Section 7.4 presents and evaluates the results of the experiment. Finally, Section 7.5 and 7.6 cover the implications and limitations of the research respectively.

7.2. Experiment Aim

The aim of the experiment is to investigate the feasibility of the proposed marking tools and fulfil the following objectives:

- To find out whether the tools enable participants to provide similar feedback in terms of quality to the feedback provided by paper-based assessment.
- To compare the time spent undertaking full marking and partial marking.
- To capture the participants’ comments and opinions about the proposed marking tools.

7.3. Experiment Design

The experiment is carried out to assess the feasibility of the marking tools. Participants did not have any information on the approaches or how the marking tools are used. Therefore, a marking guide, questions and model answers were presented to each participant before the experiments. Moreover, a brief demonstration of the tools was provided before the experiments, so that the participants could use the marking tools more effectively. Each of the
participants then provided feedback using the full-marking and partial-marking tools. The participants started the experiment by using the full marking tool. This is due to the fact that the full marking tool displays the code script to participants, and as such, it is more similar to a traditional approach of marking than the partial marking tool. Subsequently, the participants used the partial marking tool, which supports a new way of marking described in Chapter 6. The order of using the marking tools was designed such that participants could begin to see the advantages of the partial marking tool after using the full marking tool.

7.3.1. Participants

This experiment was conducted with eight markers (participants). Creswell (1998) and Connelly (2008) state that at least five participants are required to capture sufficient results in a study/experiment, as was discussed in Section 5.2.2. Five participants were expert markers while three participants were non-expert markers in this experiment. Ofqual (2014) state that expert markers must have an undergraduate degree in the subject. However, the existing literature lacks clarity and detail regarding an expert marker’s experience (in terms of how many years of experience are required). In this research, expert markers were required to have at least ten years of marking experience. Finding an expert marker is often difficult since they usually have busy schedules. Thus, the five expert participants used in the main experiment were selected based on purposive sampling, which was described in Section 3.5.2.

7.3.2. Questions

The participants provided comments on novice students’ solutions which were collected via three questions. It is worth mentioning that the participants in this experiment and the experiments in Chapter 5 were different. Thus, the same questions used in the earlier experiments (Chapter 5) were used in this experiment. The questions were presented in Table 5.1. The participants were required to provide feedback for 30 code scripts (50 code segments) using each tool (some sample scripts are presented in Appendix E). The sample data originally consisted of 165 code scripts as was highlighted in Chapter 5. However, to accommodate the busy schedules of expert markers, the code scripts were examined in terms of code segment structure, programming errors, code layout and comments to reduce the number of code scripts used in the experiment, as was described in Section 5.2.3. In such a
manner, each participant made comments for a total of 30 code scripts (50 code segments) using the full and partial marking tools separately. In addition, 17 groups were created from the 50 code segments based on their similarity measurement ratios. However, it is worth mentioning that if all code scripts (165) were displayed to the participants the total marking time spent would increase, the average marking time (initial-marking made by the marker) for a code segment would be approximately the same. In light of this argument, slightly different code segments were displayed to the participants. This was done because using similar code scripts would have made the tools re-use the participants’ comments to comment on repetitive code segments based on the CBR cycle. This would have made the results of the experiment subject to bias when evaluating the marking time spent.

7.3.3. Measurements

The main measurement of this experiment was the time spent on the full and partial marking processes. The proposed semi-automated assessment approach suggests using the CBR cycle based on segmented marking in order to provide consistent feedback. In this sense, the initial and revised marking time for repetitive code segments need to be measured. Participants check the correctness of the re-used (automatically generated) comments to provide appropriate feedback. In this context, the checking time for each code segment is referred to as the revised-marking time in this study. Furthermore, the number of automatically commented (re-used comments) code segments and the number of modified (revised) code segments were recorded in the experiment. Additionally, since each participant has a unique marking experience, the feedback quality provided by the participants can be different. Therefore, this research did not focus on the feedback quality since it is dependent on the participant’s experience.

7.3.4. Data Collection

In this experiment, data was collected through two methods. Initially, the participants commented on students’ code scripts through the marking tools. The marking time spent on code segments was measured and recorded. Secondly, a questionnaire was answered by each participant after the marking process, thus capturing the participants’ thoughts on segmented marking and the marking tools.
7.4. Results and Discussion

This section presents and discusses the experimental results based on the marking time spent, quantitative measurement and questionnaire results.

7.4.1. Marking Time

Each participant completed the marking processes successfully by providing comments for each code segment. The marking time was measured for each code segment for both marking tools. It should be noted that the first three participants (in terms of their ID number) are always referred to as the non-expert markers in the figures/tables of this section. The total time spent on each marking tool is illustrated in Figure 7.1.

![Fig. 7.1 Total time spent on each marking tool](image)

From Figure 7.1, one can see that all the participants spent less time on the partial-marking tool than the full-marking tool. Figure 7.2 and Figure 7.3 display the average initial- and revised-marking time spent for code segments on the full and partial marking tools respectively.
Figure 7.2 illustrates that in the full marking tool, each participant spent less time checking the correctness of comments for code segments than they did making the initial marking. On average, participants spent around 78% less time checking the correctness of the re-used comments for each code segment through the full-marking tool than making the initial marking.

The data in Figure 7.3 shows that, as expected, each participant spent more time on the initial marking than on the revised marking when using the partial marking tool. On average, participants saved around 86% of the initial-marking time by checking the correctness of the re-used comments for each code segment through the partial-marking tool.

Participants provided comments for seventeen groups using each tool, as described in Section 7.3.2. Figure 7.4. and Figure 7.5. illustrate the marking time spent for each code segment in a single group using the full and partial marking tools respectively. The marking times spent for the rest of the groups (16 groups) are also available in Appendix C.
**Fig. 7.4** Marking time spent on group members using the full-marking tool

**Fig. 7.5** Marking time spent on group members using the partial-marking tool
From the data in Figure 7.4 and Figure 7.5, it is apparent that the marking time spent reduces between group member 1 and group member 4. In fact, one can see that the marking time spent reduces markedly between group member 1 and group member 2 in both figures. Group member 1 was commented on by the participants manually (initial marking) while the rest of the group members were commented on automatically by the tools. As such, the marking time spent reduced sharply between group member 1 and group member 2. The participants subsequently spent 76% and 83% less time on average in the revised marking using the full (Figure 7.4) and partial marking tools (Figure 7.5) respectively.

Overall, from the marking time measurement results, it can be understood that both marking tools saved participants’ time in the marking process. In addition, participants saved more time by using the partial marking tool than the full marking tool.

### 7.4.2. Quantitative Measurements

The participants’ comments were re-used to comment on repetitive code segments automatically, as described in Section 4.6. The number of initially marked, automatically marked (re-used comments) and modified (in the revise process) code segments are presented in Figure 7.6 (for the full-marking tool) and Figure 7.7 (for the partial marking tool). The participants were required to comment manually on 17 code segments since the 50 code segments used consisted of 17 groups. The manually commented code segments thus represent 34% of the total code segments.
From the data in Figure 7.6, it can be seen that on average approximately 13% of the automatically marked code segments were modified in the optional revision process by the participants using the full marking tool.

![Bar chart showing the number of marked code segments according to type using the partial-marking tool.](chart)

**Fig. 7.7 Number of marked code segments according to type using the partial-marking tool**

Figure 7.7 shows that 66% of the code segments were commented on by the partial-marking tool (automatically), while 25% of the automatically commented code segments were modified in the optional revision process by the participants.

According to the results above, the number of revised code segments using the partial-marking tool was more than the number of revised code segments using the full-marking tool. The two marking tools support different marking techniques, which were described in detail in Section 6.4. In the partial marking tool, the participants were only able to access code segments instead of the full code scripts. On the other hand, the participants could see the full code script in the full marking tool. In this sense, the participants might have provided insufficient feedback while they were using the partial marking tool. This is because they might not have been able to link between code segments in the partial marking technique, as they did not have a higher-level view of the code they were marking. As such, they might have needed to modify more comments in the partial marking tool. Furthermore, the result also indicate that the full-marking tool is more reliable than the partial marking tool in terms of providing correct and applicable comments. However, the average revised marking time spent on code segments was notably less than the average initial marking time, as highlighted in Section 7.4.1. In this manner, participants could efficiently double check the correctness of the comments during the review process. In addition, the results displayed in Figure 7.6 and 7.7 point out that participants were
able to provide consistent comments (around 92% using the full-marking tool and 84% using the partial marking tool on average). This is because they did not revise too many comments in the marking process.

7.4.3. Questionnaire Results

The following questions and statements were asked in the questionnaire to each participant at the end of the marking process. The questionnaire was divided into four parts. The first part examined the participants’ paper-based marking procedure, whereas the second part investigated the efficiency of the marking tools. The third group examined the usage area of the marking tools. Finally, the fourth group obtained the participants’ general opinions about the marking techniques and tools.

1. General Questions

**Question: What is your paper-based assessment procedure?**

**Rationale:** This question was asked to find out the participants’ paper-based assessment procedures. At the end of the experiment, the participants’ comments on the tools and their paper-based assessment procedures will be compared. The participants’ responses about their paper-based assessment procedures are presented in Table 7.1.

<table>
<thead>
<tr>
<th>Participant</th>
<th>Paper-Based Assessment Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“Syntax errors, solution method and code output.”</td>
</tr>
<tr>
<td>2</td>
<td>“Code layout, syntax errors and code output”</td>
</tr>
<tr>
<td>3</td>
<td>“Check syntax errors, correctness of the output and readability.”</td>
</tr>
<tr>
<td>4</td>
<td>“I mark code digitally using a marking scheme.”</td>
</tr>
<tr>
<td>5</td>
<td>“I check the first ten or fifteen students’ solutions before I begin the marking process. I prepare a marking scheme based on the students’ solution. I try to give more points for the similar parts in the students’ solutions.”</td>
</tr>
<tr>
<td>6</td>
<td>“I set up a detailed marking scheme. “</td>
</tr>
<tr>
<td>7</td>
<td>“I mark students’ scripts using a marking scheme. “</td>
</tr>
<tr>
<td>8</td>
<td>“I provide feedback based on a marking scheme.”</td>
</tr>
</tbody>
</table>

From Table 7.1., it is clear that the participants adopt different marking procedures. The first three participants (non-expert markers) provide comments based on the syntax errors and the outputs of the students’ solutions. From the non-expert participants’ responses, one can see that
they probably run students’ solutions to check their correctness and to provide comments on syntax errors in a digital platform. However, their marking procedures are not completely based on paper-based assessment. On the other hand, the expert participants prefer to use a marking scheme in their marking procedure. One particular participant provides feedback digitally and prepares the marking scheme based on the correct and similar parts of the students’ solutions. However, the overall responses to the question were poor since they did not provide detailed information on the participants’ own paper-based assessment procedure. For instance, their responses did not reveal whether the participants focus on segmented marking.

**Question: What kind of marking approach have you used?**

**Rationale:** Although many researchers have been used to using online assessment approaches for decades, the majority of markers prefer to provide feedback using a paper-based approach as was described in Chapter 2. This question was asked to find out whether the participants prefer to use online assessment approaches.

All non-expert participants had only used paper-based approaches prior to the experiment. In this sense, the proposed marking tools may encourage them to use computer-based assessment approaches based on formative assessment. On the other hand, the expert participants had used both paper-based and computer-based assessment approaches. This is an important point since they could compare the marking tools’ approaches they had used before with the proposed full and partial marking tools’ approaches. Thus, they could provide valuable comments and opinions on the proposed marking tools.

II. Efficiency

**Statement: If I provided tool-based feedback instead of using the paper-based approach, I could generally provide the same level of feedback.**

**Rationale:** This question was asked to find out whether the tools enable the participants to provide the level of feedback they desire.
Figure 7.8 shows that the majority of participants (63%) agree and 37% of the participants strongly agree with the statement in the context of using the full-marking tool. On the other hand, 12.5% of the participants hold a neutral view on the statement in the context of using the partial-marking tool. Furthermore, 75% of the participants agree and 12.5% strongly agree with the statement for the partial marking tool. Overall, the data in Figure 7.8. above highlights that the participants are satisfied by using the marking tools. Furthermore, from this data, one can see that the full-marking tool enables the participants to provide more similar feedback to that which is provided with their paper-based approach more than the partial-marking tool.

Statement: If I provided tool-based feedback instead of using the paper-based approach, I could generally provide the same level of feedback.

Rationale: This question was asked to find out whether the participants provide segmented marking in their marking approaches.

Figure 7.9 Providing segmented marking in paper-based approaches
Figure 7.9 shows that the majority of participants (87.5%) agree with the statement. According to the participants’ paper-based marking procedure described in Table 7.1, they do not utilise segmented marking. In light of this, the participants’ responses on this statement are not surprising. In addition, segmented marking may be time-consuming in paper-based approaches for crowded classrooms. In such cases, providing feedback for each code segment in a code script would require a lot of effort and encourage markers to provide general comments on code scripts instead of providing segmented marking. However, this could result in missing some parts requiring comments on code segments and reducing the overall quality of feedback given to students.

**Statement:** The marking tools reuse my comments and enable me to check the correctness of the generated feedback. This allows me to focus on each code segment.

**Rationale:** This question was asked to find out whether the tools encourage the participants to focus on code segments.

![Graph showing the responses to the statement](image)

**Fig. 7.10 The re-use comments in the marking process**

According to the data in Figure 7.10, it is apparent that the majority of the participants (87.5%) agree with the statement for both marking tools. This is a positive result for this research since it strongly advocates the re-use of markers’ comments by the marking tools, as described in Section 6.3. On the other hand, 12.5% of the participants hold a neutral view on the statement. According to data presented in Figure 7.10, it can therefore be inferred that the marking tools encourage the participants to focus on segmented marking.

Overall, the results of the post-marking questionnaire highlighted that both marking tools (especially the full marking tool) enable the participants to provide similar feedback to that provided through their own paper-based approaches. Furthermore, the tools encourage them to
provide segmented marking since the tools re-use their comments to automatically provide comments for repetitive code segments in different code scripts. In light of these findings, both tools can be considered more efficient than a manual paper-based approach. However, the full-marking tool can be considered to be more efficient than the partial-marking tool. This is because participants generally demonstrated stronger approval of the statements on the full-marking tool than the partial-marking tool. This is expected since the full-marking tool is more similar to traditional way of marking compared to the partial marking tool.

III. Usage Areas

**Question: Do you think these tools can be used to support formative assessment?**

**Rationale:** This question was asked to find out whether the participants would like to use the tools based on the formative assessment approach.

The participants consider that both tools (full and partial marking) can be used based on formative assessment. The aim of the formative assessment approach was described in detail in Section 2.2. The participants responses to this question are valuable since the proposed marking approaches are created primarily based on formative assessment.

**Question: In your opinion, what kind of programmer(s) may receive the most beneficial feedback to improve their programming skills through the tools?**

**Rationale:** This question was asked to find out whether the tools are suitable to provide feedback for novice programmers.

The participants believe that novice programmers would receive the most beneficial feedback through both tools. Expert programmers do not need detailed comments for their code scripts, as general comments are usually sufficient. However, novice programmers need detailed and helpful comments rather than general and brief feedback. Furthermore, since one of the objectives of this research was to provide feedback for novice programmers based on formative assessment, the participants’ responses to this question can be considered a positive and encouraging result for this research.

**Question: What educational level is the most suitable level to provide feedback for through the tools?**
Rationale: This question was asked to find out the tools’ most suitable usage in terms of education level.

The participants believe that the tools can be used both for secondary and higher education. The tools are implemented for the novice programmers and both education levels do indeed involve teaching novice and beginner learners.

Question: Would you like to use this kind of approach or tool for formative assessment in your institute?

Rationale: This question was asked to find out whether the participants would prefer to use the tools in their own institutes.

![Figure 7.11 Using of tools in the participants’ institute](image)

Figure 7.11 demonstrates that the majority of the participants (87.5%) would like to use both tools in their own institutes. This is a positive result for this research. Using these tools enables the participants to provide segmented feedback and allows the provision of more consistent, personalised and helpful feedback for novice programmers. The marking tools are also still prototypes and can be further developed in future.

Question: Which type of practice/exam is best for using the tools?

Rationale: This question was asked to find out whether the participants would prefer to use the tools in lab exercises.

Participants consider that the proposed tools are most useful in supporting lab exercises. In lab exercises, markers do not generally need to provide a numerical grade. However, they may provide numerical grades in a coursework, and must provide them in midterm/final exams.
These facts may have made the participants consider that lab exercises constitute the best environment for using the tools. However, the tools can be also used for coursework and midterm/final exams if they are improved in the future to provide numerical grades.

Overall, the results presented in this part of the questionnaire suggest that the tools can be used to provide feedback for novice programmers in lab exercises based on the formative assessment approach.

IV. Participants’ Comments and Suggestions

**Question:** Could you please provide comments on the proposed marking tools (full marking and partial marking)?

**Rationale:** This question was asked to find out the participants’ general opinions and suggestions on the proposed marking techniques and tools. Their suggestions can be used to improve the marking techniques and user interfaces of the marking tools in future.
Table 7.2 The participants’ comments and opinions

<table>
<thead>
<tr>
<th>Participant</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“Full-marking helped to see the full code, as all assessments should be done after reading all code. Furthermore, the suggestions made by the system were very helpful. Partial marking makes my side easier, but it can lead to incorrect assessments as I’m basing my assessment only on a snippet of the full code. The automation was well implemented, but would create confusions and further work in more complex snippets. Code segments were short so do not need any annotations. The tool would be helpful in more complex and longer code segments.”</td>
</tr>
<tr>
<td>2</td>
<td>“Full marking is a better way to describe and comment on the code sections. Re-using the comments and suggestions made my job easier. Partial marking is also a better way to comment on different sections but can be vague for an amateur. There is no need for using any annotations because the sections were short.”</td>
</tr>
<tr>
<td>3</td>
<td>“Full marking displayed full code which was helpful for me. Suggestions saved my time. Partial marking displayed code parts which could lead me to make incorrect comments. However, it was an effective way to make comments and saved my time.”</td>
</tr>
<tr>
<td>4</td>
<td>“Both approaches are good. However, the full approach is better than the partial approach. I always want to see the full code. The partial approach is faster than the full approach. Both tools are helpful for providing consistent feedback in a short time compared to my existing marking system. For example, my comments are not re-used in my system. However, my system has a general comment bank. I copy and paste comments from the comment bank to the comment area. Thus, both marking tools could suggest all previous comments rather than just a comment.”</td>
</tr>
<tr>
<td>5</td>
<td>“Both approaches are good ideas. However, I think the full approach is more helpful to make appropriate comments since it shows the full code. On the other hand, I edited more comments in the partial approach compared to the full approach when I checked the correctness of the re-used comments. However, the partial approach is faster and saved my time. If all previous comments were suggested instead of just a comment by the tools, the tools could be more helpful. In general, I like the tools.”</td>
</tr>
<tr>
<td>6</td>
<td>“I think both marking approaches are similar. However, the full approach was better than the partial approach. In the partial approach, I made comments for code snipes instead of the full code which is not a familiar way of marking. However, the partial approach is faster than the full approach. The suggestion part of the tools accelerates the marking process. However, I would like to have access to all previous comments, not just the nearest neighbour. Overall, I like the ideas of the full and partial marker. The percentage completion indicator is good and encouraging for the marker. The tools are a bit daunting at first sight but it is easy to learn how to use them.”</td>
</tr>
<tr>
<td>7</td>
<td>“The full marking approach is good. The full marking tool is better than the other tool since I see the full code in this approach. The partial marking tool is also good. It prompted comments and I had a look through again. The tools suggested only one comments. They could suggest more than one comment. Generally, these approaches are good ideas for providing consistent feedback.”</td>
</tr>
<tr>
<td>8</td>
<td>“Although the marking approaches are similar, the full marking approach is better than the partial marking approach. I modified more comments in the partial marking approach. However, the partial marking tool is faster than the full marking tool. If more comments are suggested in the marking process, I could provide feedback in a shorter time”</td>
</tr>
</tbody>
</table>

From Table 7.2, it can be seen that first three participants (non-expert markers) generally provided positive comments about the marking tools. However, they have the same concerns on the partial-marking approach. According to the participants’ comments, the full-marking
tool is better than the partial marking tool. They also stated that partial marking tool can result in providing unsuitable comments for some code segments. The partial marking environment may cause this concern since it does not allow participants to link between code segments in a code script in the marking process. However, participants might be able to compensate such a risk through the review process whilst retaining the efficiency gains since they are able to access the code scripts in the review process (see Figure 6.3). Overall, the non-expert participants highlighted that both marking tools are helpful for making comments based on the formative assessment approach. Furthermore, Table 7.2. shows that the expert participants’ comments are also generally positive on the use of the tools. They found that both marking approaches are innovative ideas and the percentage indicator provides encouragement in the marking process. In addition, they approve of the suggestion part of the marking tools which helps them to provide comments in a shorter time. However, they have some concerns relating to both tools which is that they would like to see all the previous relevant comments in the suggestion area of the tools. This is so they could accelerate the marking process by choosing the most suitable suggested comment. This may also serve to enhance the consistency of the feedback they provide. They also indicated that although the partial-marking tool is faster than the full-marking tool, they prefer the latter over the former. This could be due to the fact that they had to modify more comments in the partial-marking tool than in the full-marking tool.

7.4.4. Findings

This research advocates a semi-automated assessment approach based on the CBA cycle and examines how consistent and personalised feedback can be provided through the segmented marking. The research also explores whether participants provide similar feedback to that provided using their own paper-based approaches.

Assessment approaches have advantages and disadvantages which were described with detail in Section 2.3. Many researchers support full-marking approaches since they enable the provision of instant and consistent feedback (Kurnia et al. 2001; Choy et al. 2007; Wang et al. 2007). However, researchers have concerns about the semi-automated assessment approaches since markers do not provide instant feedback. While the provision of instant feedback is a key advantage of the full-marking approaches, markers can provide personalised and detailed comments using the proposed semi-automated assessment approach. In addition, markers can also save their time and provide comments in a quick manner in the proposed
assessment approach compared to the paper-based marking approaches. Even if markers make incorrect comments for many code segments, they could still provide consistent feedback by using the ‘save all’ button in the marking tools, as described in Section 6.6. In such a way, they can modify the comments, and all incorrectly commented code segments can then be correctly commented. Thus, this can save the markers time as well as provide consistent feedback.

In full-automated assessment approaches, human markers do not provide any comment as the system provides the feedback in full. However, human markers provide comments in this proposed semi-automated assessment approach. In this case, feedback quality depends on the markers’ experience. In this experiment, the expert participants’ comments were examined in terms of meaningfulness and usefulness, and are presented in Appendix D. Overall, the participants comments were similar. They mostly made comments on the code structures, logic errors and explained the code and the code layout. In addition, some of the participants made a few suggestions for the students. However, the non-expert participants generally provided more superficial comments (which are also available in Appendix D) since they have little experience of marking. This demonstrates that the feedback quality completely depends on the participant’s experience rather than the proposed semi-automated assessment marking tools.

Since this research focuses mainly on providing consistent feedback in order to improve novice programmers’ programming skills, assessing the quality of feedback is not a research objective. However, even if the participants’ feedback was displayed to the novice programmers (students) who provided solutions in order to survey their opinions about the quality of feedback, their opinions could be biased or they may not provide sufficient comments on the quality of feedback. For example, a feedback may have high quality and a novice programmer (student) may think that the quality of feedback is low. Thus, the generated feedback was not displayed to the students in this research. In addition to the aforementioned arguments, according to one of the findings highlighted in the previous paragraph, the quality of the feedback also depends on the marker’s experience.

The participants’ comments are re-used for similar code segments with in same group. However, care is needed while making judgement that may appear to contradict judgements in other places. For example, even if two code segments are placed into the same group, the re-used comment may not be suitable be used for both of them. In this sense, the marking system may require the approval of the participant about the correctness of the automatically re-used comments for each code segment. Furthermore, even if the participants reviewed the
automatically marked code segments in the experiment, the system could have enforced the participants to approve the correctness of re-used comments.

The proposed marking tools also enabled the participants to make annotations. However, the participants generally used between one to a maximum of three annotations, which is a disappointing result. This may due to the annotation toolbar not being user-friendly. In this experiment, the participants mainly preferred to use the comment area and made comments as they intended to. Furthermore, they were generally satisfied with the generated feedback although they did not prefer the use of annotation. In addition, after the questionnaire was carried out, one of the participants stated that “the code scripts were short answers and therefore I did not need to make annotations. I could make annotation if the code scripts were not short”. This participant’s comment is valuable in the context of this research. This is because this research aims to provide feedback based on formative assessment for novice students who generally provide short answers. As such, this approach can also work well without annotations. However, the user interface of the annotation may still need to be re-developed to make it preferable and more user-friendly to markers.

7.5. Implications

This research can contribute towards providing feedback using the traditional way of marking and the partial marking techniques through CBA systems. Additionally, consistent and personalised feedback can be provided thus saving significant time.

The proposed CBA system can be developed beyond the static assessment. For example, the system can be extended with the use of dynamic assessment. In such a manner, participants could also provide comments on the correctness of the output. In addition, the CBA system can be expanded to enable participants to provide numerical grades, which would allow the system to be used for midterm/final exams and assignments etc.

More comments can be presented in the suggestion pane instead of only the nearest neighbour’s comment. This would allow participants/markers to make more comments which may accelerate the marking process compared to the current versions of the marking tools.
7.6. Limitations

One limitation of the experiment may be related to the use of non-expert markers. This is because they do not possess much experience on marking and therefore their comments on the marking tools could be biased.

Another potential limitation is the small sample size used in the experiment. However, this was mitigated by the use of adequately expert participants, as discussed in Section 7.3.1.

An additional potential limitation is that the manual marking time spent is not directly compared against the marking times in the full and partial marking tools. The measurement of manual approach efficiency is subjective, and would require a significant sample size, and as such has been considered outside the scope of this research. Instead, since two different marking approaches are created, the marking time spent based on the two marking approaches are compared.

The annotation toolbar also presented a limitation. The toolbar is a plugin that is added to the bookmark area in web browsers. Participants needed to add the annotation bookmark into their own web browsers for the experiment. However, most participants preferred to use the researcher’s machine. One of the participants took part in the experiment remotely by connecting to the researcher’s machine over TeamViewer. Unfortunately, the use of TeamViewer reduced the internet connection speed during the experiment. However, this limitation did not significantly affect the overall outcome of the research.

7.7. Conclusion

The full and partial marking tools were evaluated in this chapter. The results show that the participants saved more time using the partial marking tool. However, the participants modified more comments while checking the correctness of the re-used comments in partial marking tool. Nevertheless, the participants were able to provide consistent and personalised feedback using both marking models, since both models adopted the CBR cycle. The participants were also satisfied with the generated feedback, which indicates that they generally provided comments using the tools as if they made them using their own paper-based assessment approach. In addition, the participants stated that they preferred the full approach to the partial approach since the former displays the code scripts in the marking process in a similar way to the traditional way of marking. This is an expected result since humans often prefer similar or
familiar methods to their own in any platform. The next chapter presents the conclusions of this research and suggests future work.
CHAPTER 8
Conclusion and Further Work

8.1. Introduction

This research has proposed a new way to provide feedback for novice programmers based on formative assessment. The proposed method suggests using both human and computer interactions to provide consistent feedback and reduce markers’ workload based on a semi-automated assessment environment. This chapter provides a review and a summary of the thesis (Section 8.2). Section 8.3 summaries the contributions. Finally, Section 8.4 outlines the limitations of this research and Section 8.5 proposes future work.

8.2. Thesis Review and Summary

Numerous researchers have stated that human assessors should be relied upon in the marking process to make the feedback more beneficial for students (Suleman, 2008; Robinson and Carroll, 2016; Tung et al., 2013). Thus, this research proposes a semi-automated assessment approach to provide more detailed, consistent and helpful feedback in the form of a formative assessment. Most of the semi-automated assessment approaches have focused on providing feedback based on dynamic assessment. Human assessors are then relied on to improve feedback based on static assessment as highlighted in Chapter 2. However, since human markers provide feedback after the dynamic assessment for each code script, this could result in making the feedback process inconsistent. In addition, these assessment systems tend to provide general comments for a programming solution (code script) instead of specific parts of the code script (segmented).

This research explores how detailed, personalised and consistent feedback can be provided for novice students’ code scripts. Discussions on the background of the assessment and the potential CBA assessment approaches have been described in a literature review study in Chapter 2. The research questions that guided the research are as follows:

1. Can a semi-automatic assessment system be used to give personalised and segmented (structural) feedback?
2. Can a semi-automatic assessment system be used to alleviate assessors’ workloads in terms of marking?
3. Can semi-automatic assessment systems provide consistent feedback on students’ exercises?

A new semi-automated assessment framework has been proposed in this research that can help human assessors to provide feedback with the aid of computers. The framework consists of a new marking approach that provides feedback for novice programmers’ code scripts based on a new marking process model. This approach relies on the similarity among code segments from different code scripts. It focuses on providing feedback based on the code segments in order to make the feedback more detailed. Marking of each code segment separately in a code script is done via a new marking technique called “segmented marking”, which has been described in Chapter 4. In this process, code scripts are parsed to obtain code segments in the segmentation process, which refers to first objective of this research. Generic rules are applied to increase the similarity between code segments in the codifying process. Then, a similarity measurement technique is applied and similar code segments are grouped in single groups in the grouping process, which fulfils the second objective of this research. In the marking process, human markers initially provide comments for a code segment from each group. Then, the markers’ comments are re-used for similar code segments based on the CBR cycle, which refers to the fourth objective of the project. Thus, the marker’s workload is reduced (research question 2) and more consistent feedback is provided (research question 3). Furthermore, the human marker is able, through this process, to provide personalised and segmented feedback (research question 1), which fulfils the third objective of this research. According to the marking process model, each marked code segment is reviewed by the human marker after each code segment is marked. Then, the reviewed code segment is stored in the case-base. In this case, the marker ensures the correctness of the comments used for the code segment. The stored code segment can be automatically re-used to comment on different code segments in the future which accelerates the automation of the marking process and enables provision of consistent feedback.

A feasibility study on the segmented marking technique has been carried out in Chapter 5, thus meeting the requirements set in the fifth objective of this project. Pilot and main experiments have been carried out to measure the feasibility of the marking technique. The study has revealed that the marking technique is feasible and found the participants in the experiments to be more focused on code segments than code scripts when providing feedback. In light of this,
it was decided to use two different marking styles, namely full-marking and partial-marking styles based on the segmented marking. The full-marking style refers to the traditional way of marking. The marker sees the code script and provides comments for each code segment in the code script. On the other hand, in partial-marking style, the marker sees only a code segment in the marking process and the marking system then combines the marker comments to generate feedback for code scripts. That is, the marker does not see the whole code script in the partial marking style. The two marking styles have been developed in Chapter 6.

The design requirements for the full and partial marking tools’ user interfaces have been provided in Section 6.4 based on the findings from the feasibility study. The full and partial marking process models have been developed based on the proposed (initial) marking process model in Chapter 4. Furthermore, the case definition, generation and usage have also been described in Section 6.5.2. Finally, the development of the prototype marking tools based on the marking models have been described and discussed in Section 6.6, which refers to the sixth objective of this project.

The evaluation of the prototype marking tools (which refers to seventh objective of this research) consists of three parts. In the first part, the average marking time spent is measured based on the initial marking and revised marking. The second part focuses on the quantity measurements. The number of modified code segments in the revised marking are presented and discussed. Finally, the thoughts of the participants, who used the prototype marking tools, are captured using a questionnaire. Details of this evaluation study have been presented in Section 7.4.

8.3. Outline of Contributions

This section reiterates and provides information about the contributions of this thesis. The section also provides references relating to chapters for each contribution and objective of this research.

- Through the application of Computer-Based Assessment (CBA) in the programming solution (code script) domain, an enhanced understanding of semi-automated assessment has been contributed. The core process of the proposed framework has increased the quality and efficiency of the feedback. Chapter 4 presented a discussion on the semi-automated assessment approach. This approach increased automation in the marking process based on static assessment, in contrast to all semi-automated
assessment marking systems that have been developed so far which focus on dynamic assessment in order to increase automation.

- A new semi-automated assessment framework has been proposed consisting of integration of a variety of processes. Such processes include obtaining individual code segments from code scripts (segmentation process) and finding similar code segments amongst different code scripts (grouping process). The aforementioned process also constitutes two of the objectives of this research (objectives 1 and 2), as described in Section 1.2. The framework can be used to accelerate the automation in the marking process and enable providing consistent feedback using the CBR cycle, which is one of the key objectives (objective 4) of this research. Section 4.5 has provided more information about the framework.

- A novel marking technique (segmented marking) has been developed based on the proposed assessment approach, thus fulfilling another objective (objective 3) of this research. A feasibility study of the segmented marking (objective 5) has contributed towards finding a novel way of marking, namely partial marking. The discussion on the feasibility study can be found in Chapter 5.

- Two novel marking process models have been developed, namely the full and partial marking process models. The process models describe all steps required in the marking part of the semi-automated assessment approach. The models ensure the consistency of the feedback and they are discussed in detail in Section 6.5. In addition, two prototype marking systems have been developed based on this process which fulfils objective 6 of this research.

- A new case concept has been defined based on the segmented marking. Generation and usage of a case based on the Case-Based Reasoning CBR cycle have also been described. The new case definition was explained in Section 6.5.2. The new case concept is used as part of the proposed framework which contributes to ensuring consistency and efficiency of feedback, and which has been evaluated in Chapter 7. That is, both marking systems have been evaluated by user tests and their results have been compared, which refers to objective 7 of this research.

8.4. Limitations

Although this research has succeeded in achieving its aims and objectives, there are also some unavoidable limitations, which are mentioned in this section.
8.4.1. Sample Size in the Experiments

The feasibility and evaluation studies have been conducted with a small sample size since finding an expert marker is not easy. The reason behind this is that expert markers are busy people and do not have enough time to test prototype marking tools. Thus, PhD students have been relied on in the experiments. However, their feedback can be considered to be more superficial than expert markers’ feedback. More participants including expert markers can be used in the future to provide a better generalization of the results. However, since sufficient sample size (described in detail in Section 3.6.2) is used in the experiments, the outcome of this research has not been affected by this limitation.

8.4.2. Use of Different Variable Names and Print Messages

String match is used to group components in the similarity measurement technique. However, students typically use different variable names and print messages. Although the used variable names and print messages are identical in terms of meaning, they cannot match and be put in to the same group, which results in increasing the number of groups. The message part of print messages is removed using the generic rules in the codifying process. Furthermore, the used variable names are fixed between code segments. These limitations could be solved using a code writing editor enabling drag and drop code parts like Scratch programming (Resnick et al., 2009). In this sense, each student can use the same variable names and print messages which may reduce the number of groups. However, it worth mentioning that students’ solutions were short answers and these answers mostly have the same structure in terms of code segment type. Furthermore, they generally use the same variable names and print messages and their solutions provide mostly same results since the asked question highlighted the print messages and variable names that should be used in the solutions. Due to these reasons, this limitation does not affect outcome of this research.

8.4.3. Questions to Collect Data

One or two code segments are sufficient for providing solutions to the asked questions in a code script. However, the questions could be designed such that more than two code segments could be obtained in a code script. This would allow more data to be collected and interpreted, thus enhancing the quality of the study. However, sufficient data is collected for this research in order to answer the research questions and as such, this limitation does not impact the outcome of this research.
8.4.4. Feedback for Missed Code Segment in Code script

Markers manually provide comment for each code segment in the marking process and the comments are then re-used for the similar code segments to mark them automatically. However, students can provide incorrect solution, including completely missing out (i.e. not providing a solution for) a code segment. In the review process, the marker still needs to provide feedback on such code segments. For this research, this limitation did not materialise based on the data and therefore it did not affect the outcome of this research. However, markers can also additionally check to detect those cases and deal with them if they occur in future.

8.4.5. Order of Code Segments

Although code segments may be in the same group, their order in their own code script is sometimes different. Furthermore, even if code segments are in same order in their own code scripts, the number of code segments in the code scripts could be different. Thus, the marking systems could provide incorrect feedback for similar code segments. In this case, different colours are used to provide correct feedback. For example, if the order of code segments and the number of code segments in the code scripts are same, a green colour is used for the marked code segment by the human marker and the automatically marked code segments. Otherwise, a yellow colour is used to indicate that the marker must check the comments to ensure their correctness. In addition, a red colour is used for the unmarked code segments. However, the sample data used in this research consisted of only two code segments, and code segments’ order within the same group was the same. Thus, the red and green colours were the only coloured flags used to provide correct feedback, which is a limitation of this research. However, sufficient data was obtained to answer the research questions using the two aforementioned colours. As such, the three colour system can be used for different sample data in future work to provide feedback.

8.5. Future Work

Several future research directions could be suggested in relation to this study, which could potentially offer new contributions to this research area. Some of these suggestions are presented in this section.
8.5.1. Dynamic Assessment

Novice programmers need feedback to improve their programming skills. This research develops a tool that provides comment-only feedback in the marking process based on the static assessment for novice programmers’ code scripts. However, dynamic assessment could be used to provide feedback on the correctness of the solutions and numerical grades could be used in the marking process. The marking approach can be developed to provide feedback about the correctness of the solution and numerical grades. Thus, this semi-automated assessment system can also be used for summative assessment. Moreover, while human markers do not provide any comment in systems supporting dynamic assessment, their comments are very important for this research since they are reused for similar code segments in the marking process. Thus, this research focuses on only static assessment and leaves dynamic assessment as a potential future work direction.

8.5.2. Adaptation Technique in the Marking Process

The null adaptation technique of CBR is used based on the similarity measurement in this research. Different adaptation techniques might be used to accelerate automation in the marking process. As such, one could propose future research where automation in the marking process may be accelerated by applying different adaptation techniques to this research’s similarity measurement. However, the marking time spent for each code segment is remarkably reduced using the null adaptation technique and so the marker’s workload is subsequently minimised. In this sense, different adaptation techniques are not core to the research problem of this research and can therefore be considered as a future work.

8.5.3. Programming Languages

Because of the time limit, the marking tools have been developed for the Python programming language only. Many researchers indicated that Python is considered to be the best language to teach to novice programmers, an argument which has been highlighted in Chapter 4. The number of generic rules (described in Chapter 4) could be extended, based on different programming languages such as C++, Java etc. Thus, the marking tools can be adapted to provide feedback for more code scripts written using different programming languages. Since this research intends to evaluate different marking techniques based on the segmented marking, different programming languages are not core to the research problem for this research.
8.5.4. Annotation Plugin

The annotation plugin has been used due to the time restrictions. Providing an annotation using the plugin is a long procedure and so it is not quite user-friendly. A user-friendly annotation can be designed and implemented in future research which may save users’ time. This research focuses on providing structural feedback based on segmented marking and the collected data from the experiments is sufficient for answering the research questions, even if annotations do indeed make the feedback stronger. Thus, annotations are not considered to be core to the research problem of this research.

8.5.5. Marking on Touch Screens

The marking tools enable the provision of feedback using a keyboard and a mouse. Such tools can be further developed in the future to be used for touch screens. Some touch screens allow the user the use of a pen (i.e. stylus). In this case, the user/marker can provide feedback using the stylus which is a similar method to the traditional way of marking. In addition, markers can use arrows and underline texts/lines to make annotations, which could be easier for markers to use than the annotation plugin used in this research. However, the annotations cannot be repeated even if they are provided using a stylus. The reason is that the annotated texts/lines must be identical to repeat the annotations. Since this research compares the two marking approaches through the two prototype user interfaces, it does not completely focus on the usability of the tools. As such, marking on touch screens is not considered to be a core feature or requirement of this research.

Finally, each limitation identified in Section 8.5 can be considered as a candidate for further research in future studies.

8.6. Overall Conclusion

A new semi-automated assessment approach has been developed in this project. Two novel marking process models have been implemented based on the findings from a feasibility study on segmented marking. Overall, markers (participants) were satisfied with the performance of the tools provided by the full and partial marking approaches. Furthermore, they were also pleased with the tools since they allowed them to provide more consistent and personalised feedback in a short time period compared to their traditional way of marking. The evaluation
results of the marking tools have highlighted that this research has achieved its aims and objectives outlined in Chapter 1.
REFERENCES


Clark, D. and Baillie-de Byl, P. (2007). Enhancing the IMS QTI to better support computer assisted marking. *International Journal of Distance Education Technologies*, 5(3), pp.8-23.


Sharma, K. K., Banerjee, K., Vikas, I. and Mandal, C. (2014). Automated checking of the violation of precedence of conditions in else-if constructs in students’ programs. MOOC, Innovation and Technology in Education (MITE), pp.201-204. IEEE.


Appendix A: Papers’ Abstract

This appendix provides a list of the abstracts of the published conferences papers by this research’s author.

**Abstract:** Programming languages have been an integral element of the taught skills of many technical subjects in Higher Education for the last half century. Moreover, secondary school students have also recently started learning programming languages. This increase in the number of students learning programming languages makes the efficient and effective assessment of student work more important. This research focuses on one key approach to assessment using technology: the semi-automated marking of novice students’ program code. The open-ended, flexible nature of programming ensures that no two significant pieces of code are likely to be the same. However, it has been observed that there are a number of common code fragments within these dissimilar solutions. This observation forms the basis of our proposed approach. The initial research focuses on the ‘if’ structure to evaluate the theory behind the approach taken, which is appropriate given its commonality across programming languages. The paper also discusses the results of real world analysis of novice students’ programming code on ‘if’ structures. The paper concludes that the approach taken could form a more effective and efficient method for the assessment of student coding assignments.


**Paper Title:** Increasing the Similarity of Programming Code Structures to Accelerate the Marking Process in a New Semi-Automated Assessment Approach

**Abstract:** The increased number of students (in higher education) learning programming languages make the efficient and effective assessment of student work more important. Thus, academic researchers have focused on the automation of programming assignment marking. However, the fully automated approach to marking has its issues. This study provides an approach geared towards the reduction of marking times while providing comprehensive, effective and consistent feedback on novice programmers’ code script. To assess novices’ code script, a new semi-automated assessment approach has been developed. This paper focuses on the semi-automatic assessment of programming code segments, partially explaining the increasing similarity between code segments using generic rules. The code segments referred to are ‘for’ and ‘while’ loops and sequence parts of code script. The initial results and findings for the proposed approach are positive and point to the need for further research in this area.

Paper Title: A new marking technique in semi-automated assessment

Abstract: The number of students learning programming languages in higher education and secondary schools has substantially increased, especially in the last decade. The increasing number of (novice) programmers makes code script assessment more important. Thus, this study proposes a new marking technique based on a semi-automated assessment approach. It advocates providing detailed and consistent feedback for novice programmers based on formative assessment. An experiment was carried out to check the feasibility of the proposed marking technique. The initial results and findings show that this is a potentially valuable approach.
Appendix B: Participants’ Comments/Feedback from Feasibility Study
This appendix presents the generically reusable and similar comments that were provided by
the participants to provide feedback for the code segments in the feasibility study of the
marking technique (Chapter 5).
<table>
<thead>
<tr>
<th>Comment No</th>
<th>Reusable Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Not necessary. It is not necessary.</td>
</tr>
<tr>
<td>2</td>
<td>Message in one line as expected and required. Message in one line. There are four print statements instead of one. Print message is OK. Print message is as requested.</td>
</tr>
<tr>
<td>3</td>
<td>Message should be in one line. Message will be presented in one line. Print statement is correct. Message broken into several pieces, should be in one line. Wrong print.</td>
</tr>
<tr>
<td>4</td>
<td>No comments. No comments at all. Lack of comments. Not any comments. There is no comment. No commenting on this code at all. No comments at all. No code commenting. It should have comments. Comments are requested. There are no enough comments to explain the code. Comments needed. Where are the comments.</td>
</tr>
<tr>
<td>5</td>
<td>Not many comments. Comments is insufficient. Comments did not explain very well. Not a lot of comments. More comments are required.</td>
</tr>
<tr>
<td>6</td>
<td>Good comments. Comments are OK. Lot of comments in the code which is good. Well commented code. Comments are proper for explaining the syntax.</td>
</tr>
<tr>
<td>7</td>
<td>Good use of the math module. Import math is fine. Good use of math module. Should use math.pi.</td>
</tr>
<tr>
<td>8</td>
<td>Math module isn’t used. Math module could be used. Use the math module. Doesn’t use the math module for ‘pi’. Should use math module. No math. Use math function instead of pi.</td>
</tr>
<tr>
<td>9</td>
<td>Task is not completed. No completed.</td>
</tr>
<tr>
<td>10</td>
<td>This calculation should ideally be in the ‘else’ part. Calculations should be in the ‘else’ part. Calculation is better in ‘else’. Should be after ‘else’. Incorrect logical order. This should be in the ‘else’ statement.</td>
</tr>
<tr>
<td>11</td>
<td>Wrong condition. It should be one condition. Condition is wrong.</td>
</tr>
<tr>
<td>12</td>
<td>‘else’ should be used.</td>
</tr>
<tr>
<td>13</td>
<td>The ‘if…elif..else’ should be used. Wrong structure. ‘Else’ should be used. Structure is wrong. Complicated structure. ‘Else’ needs to be here. ‘Elif’ structure not used. If…elif..else structure not used. No ‘elif’ use there. ‘Elif’ structure is not expected. Missing Else statement. Add ‘else’. Missing else.</td>
</tr>
<tr>
<td>14</td>
<td>Nice structure. Good structure. Structure is fine.</td>
</tr>
<tr>
<td>15</td>
<td>Calculation should be in the loop. Calculation is in wrong place.</td>
</tr>
<tr>
<td>16</td>
<td>Good solution. The solution is good. Good answer. This program seems to be fine. Good program. Overall good. Correct.</td>
</tr>
<tr>
<td>17</td>
<td>Could be ‘for’ loop but ‘while’ is also fine. ‘for’ loop could be used instead of ‘while’ loop. May be used for loop.</td>
</tr>
<tr>
<td>18</td>
<td>Used integer instead of float. Should be integer. It should be integer. Should be integer not float. Variables should be converted to int not float. Int not float. Inputs should be integer.</td>
</tr>
<tr>
<td>19</td>
<td>Syntax error- ‘and’ between calculations. Error using ‘and’ in the calculations. Syntax mistake-wrongly used ‘and’. Error-‘and’ cannot be used here.</td>
</tr>
<tr>
<td>20</td>
<td>Syntax error in if condition. Wrong expression. Wrong condition.</td>
</tr>
<tr>
<td>21</td>
<td>Syntax error in condition. Condition is wrong. Incorrect comparisons. Wrong loop range.</td>
</tr>
<tr>
<td>22</td>
<td>Loop structure is ok. Loop logic is used correctly. While loop looks correct. Good loop.</td>
</tr>
<tr>
<td>23</td>
<td>Wrong condition in loop. Condition part is wrong. It should start with 2 instead of 3. Wrong range.</td>
</tr>
<tr>
<td>24</td>
<td>Hi didn’t calculate the surface area. No surface area calculation. Missing surface area.</td>
</tr>
<tr>
<td>25</td>
<td>Variable names are not consistent. Different variables.</td>
</tr>
<tr>
<td>26</td>
<td>Units ‘cm’ should be included. Add units ‘cm’.</td>
</tr>
<tr>
<td>27</td>
<td>Variables are not defined. It should be defined. It is not defined.</td>
</tr>
<tr>
<td>28</td>
<td>No columns. No table. Table needs to be created.</td>
</tr>
</tbody>
</table>
Appendix C: Marking Time Spent for Group Members
This appendix provides information on the marking time spent for each group member using the full and partial marking tools. This supplement the data in Chapter 7 which provided the marking time spent for only one group (from 17 groups).
C.1 Marking Time Spent based on the Full Marking Tool

Fig. C.1 Marking time spent on Group 2 – full marking tool.

Fig. C.2 Marking time spent on Group 3 – full marking tool.
Fig. C.3 Marking time spent on Group 4 – full marking tool.

Fig. C.4 Marking time spent on Group 5 – full marking tool.
Fig. C.5 Marking time spent on Group 6 – full marking tool.

Fig. C.6 Marking time spent on Group 7 – full marking tool.
**Fig. C.7 Marking time spent on Group 8 – full marking tool.**

**Fig. C.8 Marking time spent on Group 9 – full marking tool.**
The remaining six groups consisted of only one code segment. The marking time spent by the participants on the code segment in each group varied between 78 and 92 seconds. As can be seen from the figures above, the participants’ marking times spent reduced remarkably after...
marking group member 1. This is due to the fact that group member 1 is marked by the participants manually. The participants’ comments were then re-used to mark the rest of the group members in each group utilising the automated marking approach. Hence, marking time reduced for all group members after group member 1.
C.2 Marking Time Spent based on the Partial Marking Tool

![Graph 1: Group 2 - Partial Marking Tool](image1)

**Fig. C.11** Marking time spent on Group 2 – partial marking tool.

![Graph 2: Group 3 - Partial Marking Tool](image2)

**Fig. C.12** Marking time spent on Group 3 – partial marking tool.
Fig. C.13 Marking time spent on Group 4 – partial marking tool.

Fig. C.14 Marking time spent on Group 5 – partial marking tool.
Fig. C.15 Marking time spent on Group 6 – partial marking tool.

Fig. C.16 Marking time spent on Group 7 – partial marking tool.
Fig. C.17 Marking time spent on Group 8 – partial marking tool.

Fig. C.18 Marking time spent on Group 9 – partial marking tool.
Fig. C.19 Marking time spent on Group 10 – partial marking tool.

Fig. C.20 Marking time spent on Group 11 – partial marking tool.
The remaining six groups consisted of only one code segment. The marking time spent by the participants on the code segment in each group varied between 51 and 63 seconds in these groups.
Appendix D: Participants’ Comments/Feedback from Evaluation Study

This appendix presents the generically reusable and similar comments of non-expert and expert markers. The markers provided the comments for the code segments using the full and partial marking tools in Chapter 7.
### D.1 Participants’ (Non-Expert) Comments

#### Table D.1 Non-expert markers’ reusable comments

<table>
<thead>
<tr>
<th>Comment No</th>
<th>Reusable Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Good. Well done.</td>
</tr>
<tr>
<td>2</td>
<td>Use math function. Import math function for pi.</td>
</tr>
<tr>
<td>3</td>
<td>Need comment. Comments to explain the code!</td>
</tr>
<tr>
<td>4</td>
<td>Good comment. Very informative, good.</td>
</tr>
<tr>
<td>5</td>
<td>Comment should be before the code. Write comment first.</td>
</tr>
<tr>
<td>6</td>
<td>Wrong code. Code seems incorrect.</td>
</tr>
<tr>
<td>7</td>
<td>Structure is wrong. Check the structure.</td>
</tr>
<tr>
<td>8</td>
<td>Math library was imported but not used for the calculations. Math imported, then no need to declare pi.</td>
</tr>
<tr>
<td>9</td>
<td>Code is correct. The code seems to be correct. Works fine.</td>
</tr>
<tr>
<td>10</td>
<td>Print the output. This student does not mention print statement.</td>
</tr>
<tr>
<td>11</td>
<td>One line print. It was mentioned to print the results in one single line. Use single line for printing. Multiple print statements can be combined.</td>
</tr>
<tr>
<td>12</td>
<td>Structure is OK. Proper structure. This part of the code is fine. Good to go.</td>
</tr>
</tbody>
</table>
## D.2 Participants’ (Expert) Comments

<table>
<thead>
<tr>
<th>Comment No</th>
<th>Reusable Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Pi definition is unnecessary. Although 3.14 is a decent approximation to pi, you could use Python’s math library to get a more accurate approximation - math.pi. You have used a very rough approximation to pi - use the Python math lib for a better approximation - math.pi.</td>
</tr>
<tr>
<td>2</td>
<td>Put comment for each line. Some use of comments, although more would be useful. Please comment your code - you have made no attempt to do so here. You don't give the user a helpful or complete message. Try to use more comments in your code.</td>
</tr>
<tr>
<td>3</td>
<td>Variables names should be clear. Use better variable names.</td>
</tr>
<tr>
<td>4</td>
<td>Put print statement. Print the output.</td>
</tr>
<tr>
<td>5</td>
<td>Don’t use two prints. Should print vol and area on one line.</td>
</tr>
<tr>
<td>6</td>
<td>Well done. Good, the program prints the vol and area on one line. You are printing the surface area and volume in one line, well done!</td>
</tr>
<tr>
<td>7</td>
<td>Use math function. Math library is suggested.</td>
</tr>
<tr>
<td>8</td>
<td>Calculation should be in else statement. Put calculation in else statement.</td>
</tr>
<tr>
<td>9</td>
<td>A new variable is not necessary. You don’t need to use the variable.</td>
</tr>
<tr>
<td>10</td>
<td>Check the calculation, it seems wrong. Be careful - volume will not be calculated correctly as you have not used brackets to ensure correct precedence of operators.</td>
</tr>
<tr>
<td>11</td>
<td>Very detailed set of comments - well done! Lots of very detailed comments - well done! Good use of comments - well done!</td>
</tr>
<tr>
<td>12</td>
<td>Structure is Ok. Good use of ‘if’ ‘elif’ and ‘else’.</td>
</tr>
<tr>
<td>13</td>
<td>Loop structure seems not correct. I don’t think your columns are going to line up here! Prints three columns as expected.</td>
</tr>
<tr>
<td>14</td>
<td>Good variables. Variable names are clear.</td>
</tr>
<tr>
<td>15</td>
<td>Calculation is OK. Calculation of vol and area is fine.</td>
</tr>
<tr>
<td>16</td>
<td>Very good, although be careful with your column widths as they may not all line up. Good table presentation. Prints table correctly.</td>
</tr>
<tr>
<td>17</td>
<td>Check syntax of your code. Syntax error.</td>
</tr>
<tr>
<td>18</td>
<td>Use integer. Do not use float.</td>
</tr>
<tr>
<td>19</td>
<td>Variable names should be consistent. Use same variable names.</td>
</tr>
<tr>
<td>20</td>
<td>Not necessary. Not essential.</td>
</tr>
<tr>
<td>21</td>
<td>Why you used while loop. Could be while loop instead of for loop.</td>
</tr>
<tr>
<td>22</td>
<td>Your formula for the surface area is not quite correct - be careful! Incorrect SA calculation.</td>
</tr>
<tr>
<td>23</td>
<td>Good use of for loop. For loop structure is fine.</td>
</tr>
<tr>
<td>24</td>
<td>Good use of while loop. Your while loop structure is OK.</td>
</tr>
</tbody>
</table>
Appendix E: Sample Scripts used for Evaluation Study

This appendix presents some raw sample scripts used in the experiment described in Chapter 7.

Table E.1 Raw Sample Scripts

<table>
<thead>
<tr>
<th>Sample Script</th>
</tr>
</thead>
<tbody>
<tr>
<td># Question 1</td>
</tr>
<tr>
<td>import math</td>
</tr>
<tr>
<td>r = int(raw_input(&quot;Please enter a value for the radius: &quot;)) # Int. Not float.</td>
</tr>
<tr>
<td>h = int(raw_input(&quot;Please enter a value for the height: &quot;))</td>
</tr>
<tr>
<td>pi = 3.14</td>
</tr>
<tr>
<td>surf = pi<em>r</em>(r+(h<strong>2+r</strong>2)**0.5)</td>
</tr>
<tr>
<td>vol = pi<em>r**2</em>h/3.0 # Decimal to be safe, but it is not required.</td>
</tr>
<tr>
<td>print &quot;The surface area is: &quot;,surf,&quot;cm^2 and the volume is: &quot;,vol,&quot;cm^3&quot; # Units added to the results.</td>
</tr>
</tbody>
</table>

| r=int(raw_input('Enter radius of the base in cm ')) |
| h=int(raw_input('Enter height in cm ')) |
| pi=3.14 |
| sa=pi*r*(r+h**2+r*r)**0.5 |
| v=(pi*(r*r)*h)/3 |
| print 'The surface area in cm is:',sa, |
| print ' and the volume in cm is:',v |

| import math |
| r = int(raw_input("Please enter a value for the radius: ")) # Int. Not float. |
| h = int(raw_input("Please enter a value for the height: ")) |
| pi = 3.14 |
| surf = pi*r*(r+(h**2+r**2)**0.5) |
| vol = pi*r**2*h/3 |
| import math ## have to import the math function so we can use it later |
| pie = 3.14 ##setting up my constants |
| r=int(input("Please enter a value for r :")) |
| h=int(input("Please enter a value for h :"))##getting my inputs |
| r2=(r*r)## creating r^2 nd h^2 |
| h2=(h*h)## creating r^2 nd h^2 |
| SurfaceArea = (pie*r)+math.sqrt(h2+r2)##creating the surface area using the math funtion |
| volume = (pie*r2*h)##getting the top half of the formula |
| vol=(volume/3)##deviding by 3 |
| print "Surface Area = ",SurfaceArea |
| print "volume = ", vol ##checking the values are coming out as something |
import math
r=int(raw_input("Please enter a value for the radius: ")) # Int. Not float.
h=int(raw_input("Please enter a value for the height: "))
pi = 3.14

# A program to calculate the surface area and volume of a cone once the

r=int(raw_input('Enter radius of the base in cm '))
h=int(raw_input('Enter height in cm '))
pi=3.14 #pi is a float value.

if r<0 or h<0: #if function printing the error message if the r or h is negative
    print 'ERROR: Both the r and h values must be positive numbers!'

elif r>100 or h>100: #elif function printing the error message if r or h is over 100
    print 'ERROR: Both the r and h values must be less than or equal to 100 cm!'

else:
    sa=pi*r*(r+(h*h+r*r)**0.5) #equation used to calculate the surface area.
    v=(pi*(r*r)*h)/3 #equation used to calculate the volume.
    print sa,
    print v #printing the sa and v on one line with a friendly message.

import math
r=int(raw_input("enter radius of cone: "))
h=int(raw_input("enter height of cone: "))
pi=3.14

surfacearea=pi*r*(r+(h**2+r**2)**0.5)
volume=pi*r**2*h/3

if r<0 or h<0:
    print "ERROR: Both the r and h values must be positive numbers!"

elif r>100 or h>100:
    print "ERROR: Both the r and h values must be less than or equal to 100 cm!"

else:
    print "The surface area is: %2.2f cm2 and the volume is: %2.2f cm3"%(SurfaceArea,vol)
    print "The surface area is: ",surfacearea," cm2 and the volume is: ",volume," cm3"
#user has entered a value
# import the math module
import math
# ask the user to enter a value for the radius of the base
radius=int(raw_input("Please enter a value for the radius of the base\n"))
# ask the user to enter a value for the height of the cone
height=int(raw_input("Please enter a value for the height of the cone\n"))
if radius<0 or height<0:
    print "ERROR: Both the r and h values must be positive numbers!"
elif radius>100 or height>100:
    print "ERROR: Both the r and h values must be less than or equal to 100 cm"
else:
    # the equation for calculating the surface area of the cone
    S_Area = ((math.pi)*(radius))*(radius+math.sqrt(height**2+radius**2))
    # the equation for calculating the volume of the cone
    Volume =((math.pi*radius**2*height)/3)
    # Allows the user to see a displayed result for the surface area and volume
    # in one line
    print "The surface area is: \n\nS_Area, cm2 and the volume is: \n\nVolume, cm3."

import math
# Imports the math module
r=int(raw_input("Please enter the value of the radius of the base: "))
h=int(raw_input("Please enter the value of the height of the cone: "))
# Asks the user to input the values for the radius of the base,
# as well as the height of the cone
if r<0 or h<0:
    print "Both the r and h values must be positive numbers!"
# Checks if the values of r and h are negative numbers
elif r>100 or h>100:
    print "Both the r and h values must be less than or equal to 100 cm!"
# Checks if the values of r and h are less than or equal to 100 cm
else:
    surfacearea=(math.pi*r)*(r+math.sqrt(h**2+r**2))
    volume=(math.pi*(r**2)*h)/3
    # Calculates the surface area of the cone using the user input as well as pi
    # Calculates the volume of the cone using the user input as well as pi
    print "The surface area is: \nsurfacearea, cm2, and the volume is: \nvolume, cm3"
# Prints a user friendly message telling them the surface area and volume

import math
# import the math module
# print the column headings for the table
print "Height/Radius Surface Area Volume "
# print the line underneath the column headings
print "="*30
# set the starting count equals to 2 as the question asks for it
count = 2
# insert a while loop where the count can only go until 10
while count < 11:
    # insert the formula for calculating the surface area and volume with
#replacing all variables as count due to the height and radius being equal in this question
S_Area= (math.pi)*(count)*(count+math.sqrt(count**2+count**2))
Volume= (math.pi*count**2+count**2)/3
#format the results and align the decimal points with 2 decimal places
print "%2d %7.2f %7.2f"%(count, S_Area, Volume)
#set the count to be increased by 1 each time
count = count + 1

import math
##we print the heading of each column as followed:
print "Height/Radius","Surface Area ","Volume"
##all possible integers from 2 to 10 is listed by using range
for count in range(2,11):
    ## formula for surface area and volume is entered, named SA and V
    SA=pi*count*(count+math.sqrt(count**2+count**2))
    V=pi*(count**2)*count/3
    ##Answer for each column will be printed using percentage sign. The integer on
    ##the right of each percentage shows the number of spaces it will have and the
    ##decimal shows the number of decimal places it will have, in which this case
    ##it will be to 2 decimal place.
    ##the variable name in brackets at end of line shows which values are to be
    ##see in each column. Here it is in order of count, surface area and volume.
    print "%10d %15.2f %10.2f"%(count,SA,V)

import math #For later use of math.sqrt
pi=3.14
#First row in table
print "Height/radius Surface area Volume"
#Line separating first row and results
print "="*30
#Loop
for r in range(2,11): #i being used as variable for height/radius
    h=r #So the calculations will work
    Area=pi*r*(r+(math.sqrt(h**2+r**2)))
    Volume=(pi*r*r*h)/3.0
    print "%5i %25.2f %20.2f"%(r,Area,Volume)

import math #Imports the math module
print "Height/Radius Surface Area Volume" #Creates/prints the table headers
print "="*30 #Creates/prints the table separator
for i in range(2, 11): #i being used as variable for height/radius
    surfArea=(i**3+(i**2)*math.sqrt(i**2+i**2)) #Equation for surface area
    volume=((i**3+2*i*i)/3) #Equation for volume
    print "%8d %20.2f %15.2f"%(i,surfArea,volume) #Prints the height/radius, surface area
    #and volume at certain indentations in
    #to create the look of a table
n=2
while n<11:
    t=(math.pi*n)*(n+(math.sqrt((n*n)+(n*n))))
    b=(math.pi*(n*n)*n)/3
    print "%-2d %20.2f %17.2f"%(n,t,b)
    n=n+1

# math library
import math
print "Height/Radius  Surface Area  Volume "
print "=================================
for r in range(2,11) :
    sa=(pi*r*(r+(math.sqrt((r**2)+(r**2)))))
    v=(pi*(r**2)*r)/3.0
    print "%1d %24.2f %15.2f"%(r,sa,v)